Implementación de un algoritmo para eliminación de ruido impulsivo en imágenes y análisis comparativo de tiempos de respuesta bajo arquitectura GPU y CPU

Sergio Alberto Cano Ocampo

Neilor Esteban Díaz López

Trabajo de grado para optar al título de Especialista en ingeniería de software.

Jaime Alberto Echeverri
Asesor Temático

Jesús Andrés Hincapié Londoño
Asesor Metodológico

Universidad de Medellín
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín
2013

CONTENIDO

INTRODUCCION 6

1. PRELIMINARES 7

1.1. PLANTEAMIENTO DEL PROBLEMA 7

1.2. OBJETIVOS 11

1.2.1. OBJETIVO GENERAL 11

1.2.2. OBJETIVOS ESPECÍFICOS 11

1.3. JUSTIFICACIÓN 12

1.4. MARCO TEÓRICO 14

1.5. RESUMEN 17

1.5.1. RESUMEN 17

1.5.2. ABSTRACT 18

2. ESTADO DEL ARTE 19

2.1. DESCRIPCIÓN DEL PROYECTO 23

2.1.1. INFORMACIÓN DE LOS INTEGRANTES 23

2.1.2. ARTEFACTOS DEL PROYECTO 23

2.2. HIPÓTESIS DEL TRABAJO 24

3. MARCO METODOLÓGICO 25

3.1. ANALIZAR CÓDIGO EXISTENTE SOBRE ALGORITMOS DE ELIMINACIÓN DE RUIDO IMPULSIVO 25

3.1.1. ELEMENTOS DE DISEÑO Y ARQUITECTURA 30

3.2. CODIFICAR EL ALGORITMO PARA SU FUNCIONAMIENTO EN ARQUITECTURAS GPU Y CPU 34

3.3. ELABORAR PRUEBAS SOBRE EL ALGORITMO IMPLEMENTADO. 39

3.4. ESTABLECER COMPARATIVOS EN TÉRMINOS DE TIEMPOS DE RESPUESTA Y PROCESAMIENTO DEL ALGORITMO BAJO AMBAS ARQUITECTURAS (GPU Y CPU) 44

ANEXOS 50

GLOSARIO 54

CONCLUSIONES 56

BIBLIOGRAFÍA 58

INIDICE DE IMÁGENES

Figura 1-1.Gráfico diferencia CPU y GPU 7
Figura 1-2. Gráfico Ascenso Tecnológico Procesadores (Ley de Moore) 11
Figura 1-3. Diagrama de bloques de imágenes en frecuencia 13
Figura 1-4. Ruido impulsivo en imágenes 14
Figura 1-5. Función en base radial 14
Figura 3-1. Comparación resultados algoritmos 25
Figura 3-2. Método vs tiempo. Imagen 512 x 512 utilizando Figura 3-1 (A) 26
Figura 3-3. Pseudocódigo de la solución 27
Figura 3-4. Diagrama de clases de la solución 29
Figura 3-5. Clases asociadas a la librería Cudafy 30
Figura 3-6. Clases utilizadas de .Net Framework 4.5 30
Figura 3-7. Clase que contiene los métodos para aplicar el algoritmo en CPU 30
Figura 3-8. Clase que contiene los métodos para aplicar el algoritmo en GPU 31
Figura 3-9. Historia de los lenguajes para computación GPU 33
Figura 3-10. Interfaz aplicación pdi_CPU_GPU 35
Figura 3-11. Flujo procesamiento CPU a GPU 36
Figura 3-12. Flujo procesamiento CPU a GPU, Carga y Ejecución 37
Figura 3-13. Flujo procesamiento GPU a CPU 37
Figura 3-14. Imagen error TDR 38
Figura 3-15. Interfaz aplicación pdi_CPU_GPU modificación Umbral 41
Figura 3-16. Vista información archivo LogResultadosPDI.csv 41
Figura 3-17. Vista información tabulada en ResultadosPDI.xslx 42
Figura 3-18. Resultados Equipo 1, Tiempo CPU versus Tiempo GPU 45
Figura 3-19. Resultados Equipo 2, Tiempo CPU versus Tiempo GPU 47

INIDICE DE TABLAS

Tabla 1-1. Características CPU - GPU 8
Tabla 1-2. Funciones de base radial típicas 15
Tabla 2-1. Comparativa investigaciones 18
Tabla 3-1. Conjunto 1 Imágenes de Prueba 39
Tabla 3-2. Conjunto 2 Imágenes de Prueba 40
Tabla 3-3. Componentes hardware equipos de prueba 42
Tabla 3-4. Resultados pruebas en equipo 1 44
Tabla 3-5. Resultados pruebas en equipo 2 46

INIDICE DE ANEXOS

Anexo 1. Inventario de capacidades de investigación 50
Anexo 2. Cronograma de Actividades 51
Anexo 3. Diagrama de secuencia ProcesamientoDigitalCPU 52
Anexo 4. Diagrama de secuencia ProcesamientoDigitalGPU 53

INTRODUCCION

En el campo de visión artificial, el procesamiento de imágenes comprende el análisis y transformación de las imágenes de un estado a otro. Dependiendo de los escenarios, los estados iniciales de las imágenes pueden ser diversos. Sobre el conjunto de estados existente se identifican imágenes que presentan ruido, siendo el objetivo del análisis eliminar este defecto siempre manteniendo los detalles de la imagen. (Yu, Qi, Sun, & Zhou, 2010)

El ruido en las imágenes es una variación de información sea en brillo o color en una imagen, siendo esta variación aleatoria, y usualmente no deseada (Radhika & Padmavathi, 2010). La CPU siendo un dispositivo hardware fundamental para el procesamiento, en el diseño actual ha venido adoptando funciones comunes encontradas en la computación GPU, como el incremento en núcleos de procesamiento y hardware multihilo. Los dispositivos GPU han evolucionado más rápidamente debido a los ciclos tan cortos en su diseño a diferencia de la CPU, la alta disponibilidad de los dispositivos GPU y su bajo costo las presentan como una opción clave para procesamiento de información. (Association for Computing Machinery, 2008)

El proyecto se enfoca en la implementación de un algoritmo de eliminación de ruido el cual será ejecutado en diferentes arquitecturas, específicamente CPU Y GPU.
La CPU es comprendida como la unidad lógica de procesamiento y la GPU es definida como un dispositivo de apoyo al procesamiento.
En cuanto a las arquitecturas en el caso de la CPU, debe minimizar la latencia (tiempo al acceder a un recurso) en cada hilo.
En el caso de la GPU, elimina la latencia (tiempo al acceder a un recurso) con computación desde otros conjuntos de hilos. (Woolley, 2012)

La información de resultados en tiempos de respuesta sobre el procesamiento en las diferentes arquitecturas (CPU y GPU) será presentada y analizada.

1. PRELIMINARES

1.1. PLANTEAMIENTO DEL PROBLEMA

La tecnología ha dado pasos agigantados en la última década, desde la masificación del internet, la aparición de las redes sociales, los celulares inteligentes con numerosas y variadas aplicaciones, hasta los más potentes dispositivos o hardware, como por ejemplo los procesadores de varios núcleos, los dispositivos de un alto poder de almacenamiento de datos, los cuales esencialmente redujeron de tamaño. De igual forma, es fácil encontrar imágenes de gran calidad, gracias a cámaras de muchos pixeles o teléfonos inteligentes capaces de realizar esta operación, sin embargo, ocupan un significativo espacio en disco, características que inciden en el procesamiento de éstas por parte de la unidad central de procesamiento CPU.

En numerosas ocasiones, se ha encontrado que al obtener una imagen digital desde cualquier medio, éstas presentan algunas imperfecciones o ruidos (cualquier componente en la imagen no deseado), como el impulsivo, que se define como aquella imagen que contiene puntos blancos y negros no pertenecientes a la imagen como tal. (Boncelet, 2009)

Para realizar eliminación de este tipo de ruido se debe utilizar filtros, tal es el caso documentado por Bogdan Smolka, quien en busca de eliminar estas contaminaciones en imágenes a color, utiliza una técnica de filtrado adaptativo bastante atractiva; este nuevo filtro supera algunos filtros existentes además de poder ser utilizado en diferentes aplicaciones. (Smolka, 2010)

Es considerablemente significativo identificar fácilmente las diferencias entre CPU y GPU, tema ampliamente discutido en los últimos años, debido a la enorme cantidad de datos que se generan a través de internet, un sistema de información o simplemente una aplicación. Gracias a los avances tecnológicos, Victor W Lee†, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim†, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal y Pradeep Dubey se dieron a la tarea de iniciar un riguroso análisis entre CPU y GPU, concluyendo que la brecha entre rendimiento GPU y CPU está mucho más cerca de lo que se pensaba (2.5) debido a diversos factores que pueden influir de manera determinante, para CPU por ejemplo:

- Optimización de código,
- Reorganización de accesos a memoria
- bloqueo de caché.

Para GPU se tiene:

- Minimización de sincronizaciones globales.
- Uso de buffer compartidos.

(Brodtkorba, y otros, 2010)

La CPU, es una unidad central de procesamiento, considerada como un elemento esencial para un computador, ofreciendo características necesarias para la interacción y ejecución de cualquier sistema (capaz de interpretar instrucciones y procesar datos). La GPU por su parte, es también un dispositivo de procesamiento, pero especializado en gráficos, para soportar aplicaciones como videojuegos.

Por otra parte, Kevin Krewell, expone que, arquitectónicamente, la CPU está compuesta por unos cuantos núcleos (los procesadores utilizados en pruebas de rendimiento de la presente investigación, tienen 4 núcleos y 8 hilos de procesamiento) con una considerable cantidad de memoria caché, manejando algunos subprocesos de software a la vez (hilos de procesamiento). En contraste, una GPU está compuesta por cientos de núcleos (las tarjetas Geforce utilizadas cuentan hasta con 384 núcleos CUDA) que pueden administrar miles de hilos de procesamiento simultáneamente. (Krewell, 2009)

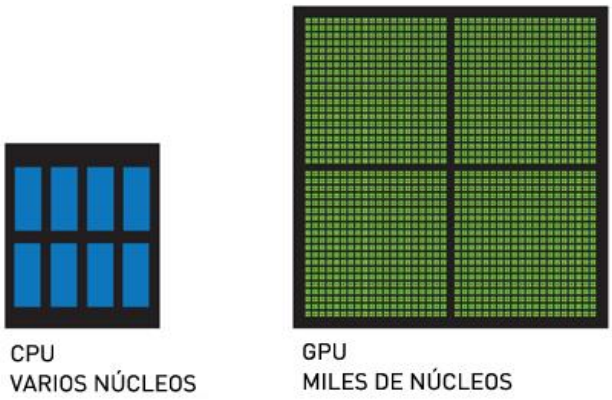


Figura 1-1.Gráfico diferencia CPU y GPU.

Tomado y modificado de ((NVIDIA Corporation, 2013))

Para identificar fácilmente las diferencias entre CPU y GPU se presenta el siguiente comparativo (tomando como ejemplo el hardware a utilizar para la obtención de tiempos de respuesta para cada arquitectura.):

Característica	CPU	GPU
Propósito	Diseñada para una amplia variedad de aplicaciones. Proporciona tiempos de respuesta rápidos a una sola tarea.	Diseñada para aplicaciones de renderización y otros gráficos con alto grado de paralelismo.
Arquitectura	Von Neuman (CPU conectado a una memoria principal, a la que se accede a través de un sistema de buses.)	Modelo circulante (facilita el procesamiento en paralelo y la segmentación)
Núcleos – Número hilos de procesamiento	Cantidad limitada (Los procesadores utilizados cuentan con 4 núcleos de procesamiento y 8 hilos de ejecución)	Gran cantidad de núcleos (se utilizan 2 tarjetas gráficas con 384 núcleos CUDA) obteniendo miles de hilos de ejecución.
Velocidad de reloj	Se refiere a la velocidad con que se ejecuta una instrucción, entre más rápido el reloj, mas instrucciones por segundo se ejecutan. Para los procesadores que se utilizaran en la ejecución del algoritmo, la velocidad es de 2.3 GHz	Para las tarjetas gráficas, se tiene una velocidad de 500 MHz (Geforce 640M) y 902 MHz (Geforce 630M)
Memoria	Generalmente maneja dos tipos principales de memoria, la memoria RAM (se cargan datos y programas) y la memoria caché (mucho más rápida, se guardan los datos más utilizados por el procesador). Los procesadores a utilizar, cuentan con una memoria Ram de 8 GB y una memoria cache L2 de 6 MB.	La GPU tiene algunos tipos de memoria adicionales, tales como: Code/Instruction Memory: (guarda que instrucción debe ser ejecuta) Shared Memory: Almacena los datos que son utilizados entre diferentes bloques. Global Memory:Memoria total de la GPU
Otras diferencias	- Rápida sincronización de operaciones - Eficiente registro operaciones SIMD (instrucción simple, multiples datos).	-Especializada en cálculos en coma flotante, así como videojuegos.

Tabla 1-1. Características CPU - GPU

En el escenario particular, relacionado a eliminación de ruido en imágenes, a pesar de la eficacia del algoritmo para eliminar ruido impulsivo en imágenes respecto al tiempo de respuesta del algoritmo que propone el filtro mediana, es importante también hacer uso de hardware dedicado, como el hardware GPU, para mejorar la experiencia en tiempo real en el procesamiento de imágenes en alta resolución. Namdev Sawant y Dinesh Kulkarni realizan un análisis comparativo entre algoritmos en paralelo (Sobre GPU-CUDA) y algoritmos secuenciales; el algoritmo utilizado CannyEdge detector para el procesamiento de imágenes, encontrando las bondades de la GPU. (Sawant & Kulkarni, 2011)

Muchos escenarios de visión artificial deben ser ejecutados en tiempo real, lo cual implica que el procesamiento de un único fotograma puede completarse en 30 o 40 milisegundos. Siendo el tiempo de respuesta uno de los requisitos más relevantes en este tipo de escenarios.

Para este tipo de requisitos, el hardware GPU, siendo dedicado, se ha convertido en uno de los más populares aceleradores para las aplicaciones con requisitos en tiempo real, sin embargo, requieren de algoritmos especializados para actuar de manera más eficiente. Margara Alessandro y Gianpaolo Cugola describen un nuevo algoritmo (Content-based) diseñado y construido para obrar de forma muy eficiente sobre CUDA (Arquitectura de programación de propósito general), usando las GPUs. En dicha investigación, se realiza una comparación entre el nuevo algoritmo y el algoritmo de coincidencia de Siena, conocido por su excelente eficiencia y demostrando los grandes beneficios de utilizar las unidades de procesamiento gráficos para mejorar rendimiento y velocidad sobre tareas específicas de gráficos. (Margara & Cugola, 2011)

Para este caso se aprovecha el poder de computación y la cantidad de memoria que ofrecen las GPU utilizando CUDA, realizando una implementación tanto de CPU como para GPU, para un algoritmo particular (Image Fusion).

Rob Farber elabora un libro donde documenta las experiencias de uso de la arquitectura CUDA para GPU's de Nvidia, este documento sirve como fuente de conocimiento en la parte conceptual de la programación BPU con CUDA, siendo esta poco conocida pero con potencial significativo. (Farber, 2011)

1.2. OBJETIVOS

1.2.1. OBJETIVO GENERAL

Implementar un algoritmo para eliminación de ruido impulsivo en imágenes y análisis comparativo de tiempos de respuesta bajo arquitectura GPU y CPU.

1.2.2. OBJETIVOS ESPECÍFICOS

- Analizar código existente sobre algoritmos de eliminación de ruido impulsivo.
- Codificar el algoritmo para su funcionamiento en arquitecturas GPU y CPU.
- Elaborar pruebas sobre el algoritmo implementado.
- Establecer comparativos en términos de tiempos de respuesta y procesamiento del algoritmo bajo ambas arquitecturas (GPU y CPU).

1.3. JUSTIFICACIÓN

El desempeño de los microprocesadores que potencian los modernos computadores ha continuado incrementándose exponencialmente a través de los años por dos principales razones (los transistores que son el corazón de los circuitos en todos los procesadores y chips de memoria han simplemente llegado a ser más rápidos a través del tiempo en un camino descrito por la ley de Moore):

Razón 1: Afecta directamente el desempeño de los procesadores construidos con estos transistores. Más allá, el desempeño de los procesadores actuales ha incrementado más rápido que la ley de Moore lo predice,

Razón 2: Los diseñadores de procesadores han sido capaces de aprovechar el incremento en número de transistores disponibles en los modernos chips para extraer más paralelismo desde el software. (Olukotun & Hammond, 2005)

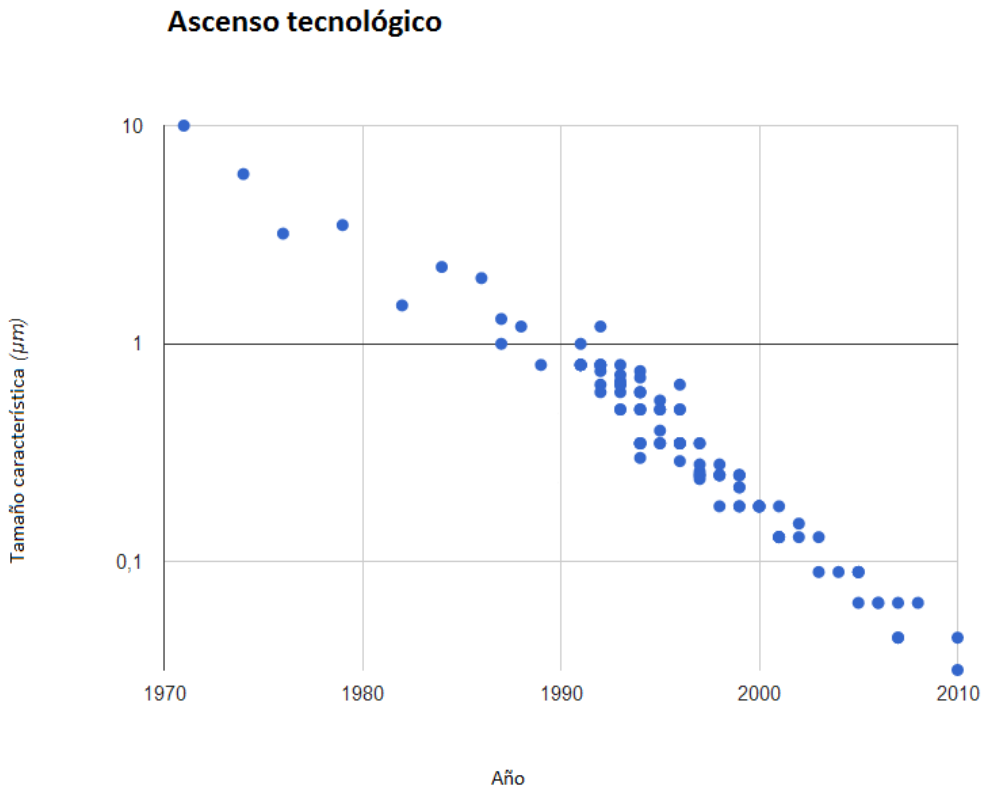


Figura 1-2. Gráfico Ascenso Tecnológico Procesadores (Ley de Moore)

Gráfico mínimo tamaño (µm, milésima de milímetro) de las características o componentes del procesador a través del tiempo. (Conocida como Ley de Moore).

Imagen tomada y modificada de (Stanford University's VLSI Research Group, 2013)

La GPU actual es un procesador versátil que constituye un punto extremo pero conveniente en el creciente espacio de las arquitecturas de cómputo paralelo multinúcleo. Estas plataformas, que incluyen GPU's, de forma progresiva los sistemas multinúcleo x86 de Intel y Amd, se diferencian entre ellos de sus diseños tradicionales de CPU al priorizar el alto rendimiento en procesamiento de muchas operaciones en paralelo sobre la baja latencia en ejecución de una tarea sencilla.

La GPU es un hardware dedicado para el procesamiento gráfico, diseñado desde su arquitectura para el procesamiento en paralelo, hasta su especialización en tareas específicas como los gráficos. Un dispositivo GPU puede entregar mayores niveles de desempeño desafiando los problemas de carga en procesos de cálculo. (Fatahalian & Houston, 2008)

La industria del software se basa en el supuesto que existe únicamente una clase de hardware (la CPU) con una interfaz estandarizada (ISA) (Set de instrucciones de arquitectura).Ésta permitió a la industria del hardware continuamente innovar en una serie de microarquitecturas que se acomodaban a la ajustada interfaz ISA. Esta acogida está siendo sustituida por programadores que buscan el máximo rendimiento absoluto o reducir el consumo de energía para recurrir a elementos de procesamiento alternativos tales como GPU's y FGPA's.

La industria del software está reaccionando al surgimiento de los sistemas multinúcleo añadiendo innovaciones a los lenguajes existentes para ayudar a expresar concurrencia y computación paralela. El uso de computación de datos en paralelo sobre GPU's, sin embargo, no podía esperar a lenguajes como C++ para ser modificados y soportar de mejor forma el paralelismo, es por esto que sistemas como CUDA fueron desarrollados para llenar el vacío. (Singh, 2011)

1.4. MARCO TEÓRICO

El mundo real es percibido por el cerebro de los seres humanos a través de unos órganos sensoriales que le envían la información, el cerebro la procesa y construye imágenes o sensaciones que le permiten interactuar con su entorno. Un proceso similar ocurre con las imágenes digitales en un computador, definidas como un arreglo bidimensional, a partir de una matriz de bits constituidas por su elemento más simple, el pixel.

Las imágenes pueden ser obtenidas a través de dispositivos conversores análogo a digital, como cámaras, celulares o escáneres, el número de bits de información de la imagen define cuantos colores representa cada pixel, es decir, en una imagen a blanco y negro, cada pixel es blanco o es negro. (Boncelet, 2009)

El capturar imágenes usando dispositivos no resulta en una imagen inalteradas, se pueden encontrar con ruidos o imperfecciones en pixeles que disminuyen notablemente su calidad, viéndose borrosas o poco nítidas. Es por esto que existen numerosas técnicas utilizadas para su mejoramiento, como por ejemplo los filtros, una imagen (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009) "en el dominio de la frecuencia o en el dominio del espacio. Los filtros en el dominio de la frecuencia se usan, principalmente, para eliminar altas o bajas frecuencias de la imagen, lo que se traduce en suavizar la imagen, o bien, realzar o detectar bordes."

Los filtros de frecuencia pueden darse en paso bajo, paso alto, paso banda. Se muestra en la siguiente figura un diagrama a bloques de imágenes en frecuencia.

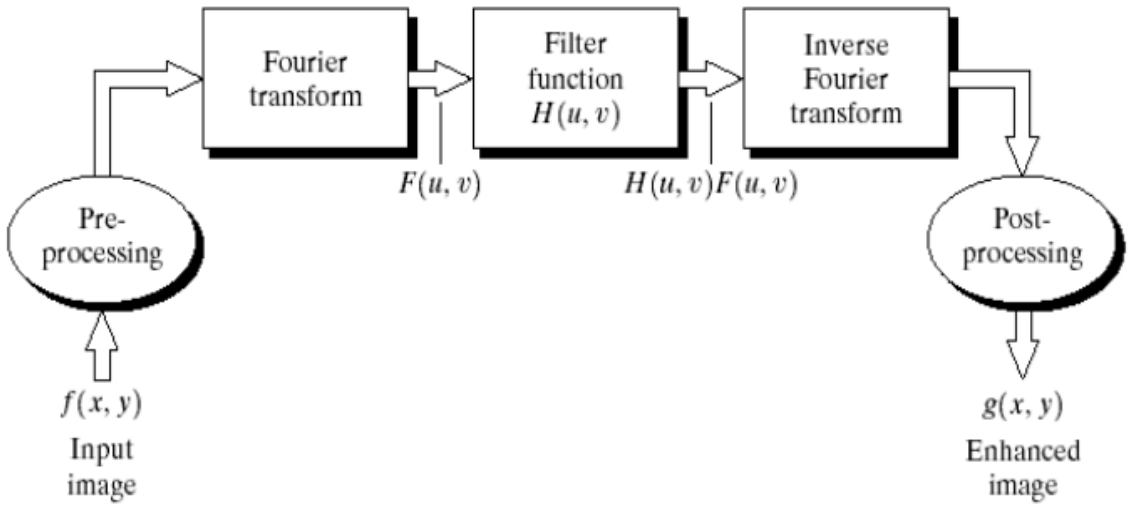


Figura 1-3. Diagrama de bloques de imágenes en frecuencia

Tomado de (Digital ImagesProcessingusing MATLAB. Gonzales 2004)

Por otra parte, el dominio espacial para el tratamiento de imágenes se basa en la manipulación directa sobre los pixeles, encontrando técnicas como la modificación del histograma, definido como un gráfico que detalla en forma global las características de la imagen; las llamadas transformaciones geométricas, que tratan problemas como la interpolación, cuando un pixel cambia de posición en una rotación (movimientos de pixeles alrededor de uno que actúa como eje).

Un tipo de ruido que se encuentra en imágenes es el impulsivo, cuya intensidad cambia bruscamente, y se caracteriza por presentar un cambio brusco en los valores de las tonalidades en píxeles aislados, según comenta (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009).

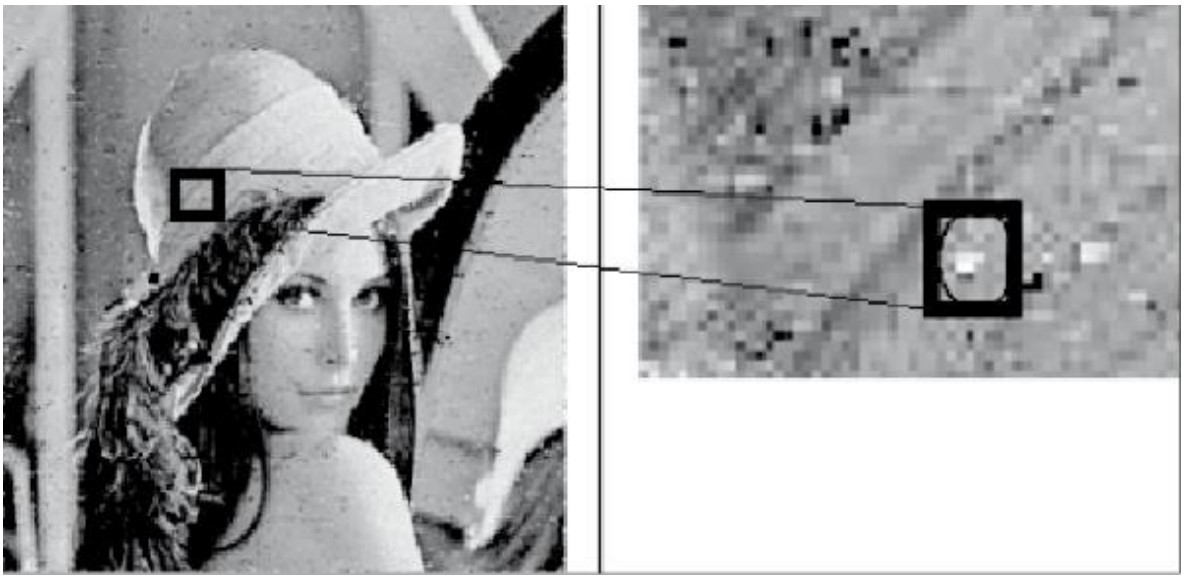


Figura 1-4. Ruido impulsivo en imágenes

Tomado de (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

En este orden de ideas, se considera la utilización de una función en base radial para la realización del presente proyecto, definida como "una combinación lineal de traslaciones de una función radialmente simétrica" (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009) de la forma:

$$s(x) = P(x) + \sum_{i=1}^{N} \lambda_i \phi(\|x_i - x\|), x, x_i \quad \in R^d$$

Figura 1-5. Función en base radial

Tomado de (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

Función	Nombre
$\phi(r) = r^2 \log(r)$	Thin Plate Spline
$\phi(r) = e^{-a^2}$	Gaussiana
$\phi(r) = (r^2 + c^2)^{\beta}$	Multicuádrica
$\phi(r) = r$	Biarmónica
$\phi(r) = r^3$	Triarmónica

Tabla 1-2. Funciones de base radial típicas

Tomado de (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

Como elemento importante, la unidad central de procesamiento CPU hereda de la arquitectura de Neumann, ejecuta el procesamiento de forma lineal, a diferencia de las GPU (Unidad de procesado de gráficos) que su esencia es el procesamiento en paralelo, lo que les ha permitido ser un dispositivo dedicado exclusivamente al procesamiento gráfico, están compuestas por un procesador de vértices, encargado de todo el proceso de iluminación y sombreado, también de un procesador de pixeles, especializado en el procesamiento y cálculo de pixeles.

Ejemplo de sistema

Recientemente, ha surgido un concepto relacionado al procesamiento por GPU, y es GPGPU (General-Purpose Computing on Graphics Processing Units), encargado del estudio y aprovechamiento de las características y potencialidades de la GPU sobre los gráficos, de hecho, Nvidia ha desarrollado una arquitectura de cálculo paralelo llamada CUDA (Compute Unified Device Architecture), que permite realizar, desarrollar y codificar diferentes algoritmos en las diferentes tarjetas gráficas de Nvidia, tales como Geforce, ION, Quadro y Tesla.

1.5. RESUMEN

1.5.1. RESUMEN

El presente trabajo, tuvo como propósito general determinar en procesamiento digital de imágenes, tiempos de respuesta al implementar un algoritmo en diferentes arquitecturas (CPU Y GPU), utilizando interpolación a través de funciones de base radial.

Para cumplir con este objetivo, se parte de una investigación previa sobre eliminación de ruido impulsivo en imágenes, a partir de allí se plantea en base a una solución en pseudocódigo un algoritmo apropiado para la arquitectura CPU y arquitectura GPU. Sobre la arquitectura GPU se detallan las particularidades identificadas al momento de la implementación (utilizando tecnología CUDA); restricciones sobre de la plataforma y alternativas de implementación.

Consecuente a la implementación, se plantea un conjunto de pruebas con imágenes, las cuales tienen ruido del tipo sal y pimienta y de diferentes dimensiones (ancho, alto), estas pruebas buscan determinar los tiempos de respuesta en cuanto a eliminación de ruido por parte del algoritmo implementado en las dos arquitecturas.

Las pruebas en tiempos de respuesta generan resultados que son analizados, principalmente evidenciando una correcta eliminación de los pixeles ruidosos (que alcanzan los 55 mil en una sola imagen) en el caso de las dos arquitecturas, y adicionalmente el tiempo de respuesta claramente bajo (mayor rapidez en procesamiento) en la arquitectura CPU con respecto a la arquitectura GPU.

Palabras Clave: procesamiento digital imágenes, eliminación ruido, arquitectura CPU, arquitectura GPU, tecnología CUDA, implementación algoritmo, tiempos de respuesta.

1.5.2. ABSTRACT

The research had as a primary objective determine in the field of digital processing images, response times in different architectures (CPU and GPU), using interpolation trough the basis radial functions.

To achieve this objective, it start with a previous research about impulsive noise elimination in images, from there its proposed an appropiate algorithm based in a pseudocode solution, to the CPU architecture and GPU architecture.

For the GPU architecture is detailed several particularities identified at the implementation phase (using CUDA technology); platform restrictions and implementation work-arounds.

As a result of the implementation phase, its proposed a test images, which have noise salt and pepper with different dimensions (width, height), these tests seek to determine response times about noise elimination by the algorithm implemented on the two architectures.

In testing the results were analyzed, mainly showing a correct noise elimination in images (reaching up to 55 thousand noisy pixels in a image) at both CPU and GPU architectures, additionally a clearly lower response time (faster in processing) on the CPU Architecture regarding to the GPU Architecture.

Keywords: digital imaging processing, noise elimination, CPU architecture, GPU architecture, CUDA technology, algorithm implementation, response times

2. ESTADO DEL ARTE

El tratamiento de imágenes es un tema ampliamente discutido e investigado, aplicando diferentes métodos que buscan obtener el mayor provecho de las técnicas para un mejor procesamiento de la imagen, en algunos casos como el presente proyecto se trata de aprovechar al hardware, como la GPU, para encontrar diferencias sustanciales en relación del mismo proceso pero en la CPU, asunto abordado en diferentes investigaciones.

Peter Bui y Jay Brockman, presentan un interesante análisis acerca del rendimiento y precisión de un método sobre CUDA-GPU y una CPU, hallando velocidades superiores de máximo 90x por parte de la GPU, pero cuidadosamente gestionando los recursos de la memoria. (Bui & Brockman, 2009)

Es de vital importancia mencionar la investigación "Performance Evaluation of Feature Extraction Algorithm on GPGPU" , quienes a través de esta, analizan las propiedades más significativas de CUDA-GPU y de forma general todo el modelo de programación, sirviendo como una referencia importante, no obstante de las comparaciones de algoritmos paralelos y secuenciales, asunto que compete totalmente a la investigación en curso. (Sawant & Kulkarni, 2011)

Jaime Echeverri, Bell Manrique, Francisco Moreno y Alejandro Bravo mencionan un método diferente, utilizando funciones en base radial para la elaboración de un algoritmo que elimina ruido impulsivo en imágenes, método que será utilizado de forma similar en la actual investigación. (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

Como producto obtenido, resultado del procesamiento, se logra una excelente calidad visual de la imagen, sin embargo, se pone de manifiesto la necesidad de comparar la técnica con otras.

Siguiendo con los métodos, Rubio Ezequiel López, manifiesta la aparición de ruido impulsivo y Gausssian, tratados en su gran mayoría para ser eliminados por separado, el propone un método basado en una clasificación Bayesiana para la eliminación de ambos ruidos. (López, 2010)

Bogdan Smolka realiza una investigación utilizando una técnica de filtrado adaptativo, considerando que el algoritmo de detección de ruido se basa en el concepto de distancias agregadas asignadas a los pixeles. El producto obtenido supera los diseños existentes para la contaminación de ruido, además de la posibilidad de utilizar el nuevo filtro en distintas aplicaciones. (Smolka, 2010)

	Algoritmo o método para eliminación de ruido	Arquitectura de procesamiento
Performance analysis of accelerated image registration using GPGPU	bicubicinterpolation- bilinearinterpolation	GPU-CPU
Performance Evaluation of Feature Extraction Algorithm on GPGPU	Canny Edge	GPU
Images improvement using radial basis functions	Funciones base radial	GPU
Adaptive technique of impulsive noise removal in color images	adaptive filtering	CPU
Restoration of images corrupted by Gaussian and uniform impulsive noise	Bayesian classification	CPU
Parallel Image Processing Based on CUDA	histogram equalization, removing clouds, edge detection and DCT encode and decode	GPU

Tabla 2-1. Comparativa investigaciones.

Las investigaciones que se han realizado al respecto intentan de manera muy puntual eliminar el ruido que se puede generar en una imagen, utilizando diferentes técnicas y modelos, desde algoritmos conocidos hasta métodos o funciones como las de base radial, esta última se toma como base para la realización del algoritmo utilizado para la comparación del rendimiento sobre las arquitecturas en mención.

En la investigación realizada por Peter Bui y Jay Brockman se brinda una importantísima información sobre el tema, ya que realizan un análisis similar al que se pretende en el actual proyecto. Utilizan la misma arquitectura de comparación CUDA-GPU con CPU, pero se difiere en el hecho de utilizar un método poco conocido o implementado como el de las funciones en base radial para la eliminación de ruido impulsivo en imágenes. (Bui & Brockman, 2009)

En el caso particular, se busca aprovechar no solo la eficacia de un algoritmo debidamente estructurado, sino también elementos poco tenidos en cuenta con el hardware.

Fuera de los algoritmos implementados, es importante conocer las experiencias realizadas en diferentes investigaciones haciendo uso de la arquitectura CUDA. En determinados casos los algoritmos deben ser adaptados para utilizar el poder de las GPU's el cual es inherentemente computacional y paralelizado. (Igual, Mayo, Hartley, Çatalyürek, Ruiz, & Ujaldon, 2011)

Estas adaptaciones requieren un considerable esfuerzo en implementación. Sobre los escenarios expuestos en la investigación mencionada, se utilizan operaciones comunes para llevar a cabo un algoritmo que pueda dar una solución a determinado problema. Estas operaciones comunes suelen ser procedimientos recursivos, desde funciones trascendentales de aritmética intensiva a operaciones de matrices. Esta investigación supone una gran base de conocimiento para la aplicación de la investigación realizada por (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

Para el caso de Yoo, Park y Jeog se aprovecha el poder de computación y la cantidad de memoria que ofrecen las GPU utilizando CUDA, realizando una implementación tanto de CPU como para GPU, para un algoritmo particular (Image Fusion). (Yoo, Park, & Jeong, 2009)

Yang, Zhu y Pu concluyen en su investigación "Parallel Image Processing Based on CUDA" que CUDA es una alternativa importante como un método de computación, siendo más económico que una implementación en hardware para tal propósito. Igualmente mencionan la aplicación de estrategias de optimización y la búsqueda de una mejor manera a futuro en cuanto al manejo de espacios en memoria. (Yang, Zhu, & Pu, 2008)

Para el procesamiento de imágenes, específicamente en el caso de Registro de Imagen (IR), Sah, Vanek, Roh y Wasnik realizan una implementación a través de la arquitectura CUDA, pero usando OpenCL permitiendo uso a nivel multiplataforma, identificando en este investigación que puede ser usado para un registro de imagen en tiempo real para imágenes de tamaño máximo 2.000 por 2.000 en cuanto a dimensiones ancho-alto. (Sah, Vanek, Roh, & Wasnik, 2012)

Che, Boyer, Meng, Tarjan, Sheaffer y Skadron en su investigación realizada resaltan información importante para tener en cuenta la implementación sobre esta arquitectura, ya que el resultado del trabajo que realizaron, se discute sobre ventajas e ineficiencias cuando se realiza una implementación sobre CUDA, ofrecen experiencias de algunas funcionalidades que podrían facilitar el uso y la legibilidad para apoyar aplicaciones de gran densidad en su codificación. (Che, Boyer, Meng, Tarjan, Sheaffer, & Skadron, 2008)

Fowers, Brown, Cooke y Stitt proponen métricas sobre diferentes dispositivos aceleradores tales como procesadores multinúcleo, gpu's, fgpa's; resultado del análisis de estas mediciones, recomiendan la GPU en caso de procesamiento y tiempo de respuesta pero en consumo de energía es muy alto, lo anterior siempre siendo considerado de acuerdo a los escenarios expuestos en el documento de investigación. (Fowers, Brown, Cooke, & Stitt, 2012)

Por otra parte, Pulli, Baksheev, Kornyakov y Eruhimov resaltan el rápido crecimiento de las GPU's, su progresiva presencia en los dispositivos de escritorio y móviles, siendo un importante acelerador en el procesamiento de determinadas tareas. En la investigación es importante resaltar la mención especial que realizan sobre los algoritmos serial, los cuales presentan soluciones al campo de visión artificial pero se caracterizan por no funcionar de manera efectiva en GPU, a su vez que son mucho más fáciles de implementar y en algunos casos se ejecuta con mayor rapidez en la CPU, lo anterior es entre otras situaciones, los retos clave en aceleración de tareas de visión artificial donde se relaciona un sistema que tiene tanto CPU como GPU. (Pulli, Baksheev, Kornyakov, & Eruhimov, 2012)

Finalmente, en la investigación de Maitre, Lachiche y Collet se menciona la importancia de la comunidad científica sobre el uso de la GPU para propósitos mucho más amplios que para los que fue concebida inicialmente (GPGPU). Expone sobre la dificultad en la implementación y resultados no esperados en cuanto a velocidad, algunas estrategias y modelos a aplicar son expuestas y finalmente se presentan resultados de rendimiento. (Maitre, Lachiche, & Collet, 2012)

2.1. DESCRIPCIÓN DEL PROYECTO

2.1.1. INFORMACIÓN DE LOS INTEGRANTES

Los integrantes del proyecto, tienen como enfoque profesional el desarrollo de software. Actualmente la actividad laboral se dá en una empresa multinacional de origen regional, común para los integrantes. Cuenta con certificaciones en los procesos, principalmente cabe destacar la certificación CMMI 5 en procesos de desarrollo de software.

Los integrantes son egresados de diferentes universidades, pero actualmente cursan en la misma universidad.

Ver detalle Anexo 1. Inventario de capacidades de investigación

2.1.2. ARTEFACTOS DEL PROYECTO

Como artefactos producto de la investigación se tiene lo siguiente:

- Código fuente de la aplicación: proyecto Visual Studio 2010 (winforms application) haciendo uso de .net framework 4.
- Manual Configuración: documentación guía para configurar un ambiente de desarrollo, donde se pueda construir aplicaciones para dispositivos GPU.
- Manual de uso aplicación: video donde se explica funcionamiento de la aplicación.
- Documento Arquitectura: proyecto Enterprise Architect donde se presenta el diseño estructural de los componentes de la aplicación.
- Documento resultados pruebas: documento en hoja de cálculo donde se detallan los resultados de las pruebas y se presentan los gráficos para cada equipo.

2.2. HIPÓTESIS DEL TRABAJO

¿Es posible determinar que un algoritmo implementado bajo la arquitectura GPU para el procesamiento de imágenes obtiene un mejor resultado específicamente en tiempos de respuesta respecto al mismo algoritmo pero bajo arquitectura CPU?

3. MARCO METODOLÓGICO

3.1. ANALIZAR CÓDIGO EXISTENTE SOBRE ALGORITMOS DE ELIMINACIÓN DE RUIDO IMPULSIVO

Las actividades iniciales se centraron en una serie de exploraciones, averiguaciones y búsquedas respecto a los diferentes algoritmos, técnicas, investigaciones y prácticas existentes para eliminar ruido impulsivo, el cual es una labor de gran importancia en el procesamiento de imágenes digitales, a lo largo de las últimas décadas, numerosos estudios e investigaciones han hecho énfasis en este tema. El filtro de la mediana por ejemplo es una técnica clásica, usada originalmente para datos estadísticos y que representa buen rendimiento y simplicidad computacional.

El ruido impulsivo ha sido abordado desde diferentes técnicas y prácticas, (García Elías & Ramírez Cruz, 2003), por ejemplo, propusieron resolver el problema del ruido impulsivo en espectros estelares utilizando redes neuronales wavelet (RNW), para lo cual realizaron experimentos entrenando la red con casos diferentes de espectros con ruido, tomándose como entrada tales espectros para la red neuronal wavelet donde se procesa y se compara la salida obtenida contra un referente sin ruido. Posteriormente se prueba la red con espectros que no fueron proporcionados en el entrenamiento y se realiza el procesamiento para filtrar y comparar los datos con la salida deseada. Los autores reportaron que la salida propuesta por la RNW es altamente aproximada a la salida deseada, lo que implica que sus resultados son satisfactorios.

Hilario Gómez Moreno, Saturnino Maldonado Bascón, Manuel Ultrilla Manso, Pilar Martín Martín, propusieron la utilización de máquinas de vectores de soporte para la eliminación del ruido impulsivo, usando tanto clasificación como regresión. Mediante el clasificador, seleccionaron los píxeles de la imagen que son ruido y mediante la regresión obtuvieron un valor de reconstrucción de dicho píxel usando los píxeles que le rodean. Esta técnica se puede aplicar con éxito inclusive, en imágenes con alta tasa de ruido mejorando significativamente la calidad de la imagen. (Gómez Moreno, Maldonado Bascón, Utrilla Manso, & Martín Martín)

Una de las técnicas no lineales más usadas para la eliminación de ruido impulsivo es la del filtro de mediana, altamente exigente a nivel computacional, al aplicarse sobre todos los pixeles de una imagen; La técnica presentada por la investigación de Jaime Echeverry, disminuye eficientemente el ruido impulsivo en imágenes, mediante el uso de un interpolante obtenido por funciones de base radial; encontrando cuales son los pixeles ruidosos, y aplicándoles ese valor encontrado en la interpolación, generando bajos costos computacionales y una buena calidad en la imagen resultante.

Esta investigación de Jaime Echeverri Arias; Bell Manrique Losada; Francisco Javier Moreno; Alejandro Bravo; es tomada como base, donde queda manifiesta la utilidad de las funciones de base radial en la eliminación de ruido impulsivo, en la cual, se realizaron pruebas con el algoritmo propuesto y algunos métodos clásicos de filtrado no lineal, como el de la media, la mediana y el outlier, demostrando ser una técnica eficiente para grandes volúmenes de ruido y más eficaz que los métodos clásicos. Es de resaltar que, inclusive son mejores que los reportados en trabajos como (Morillas & Valentín, 2011) y (Kumar, Kumar, Kiran, & Rama Krishna, 2011) en cuanto a la relación señal a ruido de pico. El tiempo de procesamiento, es proporcional a la cantidad de ruido presentada en cada imagen seleccionada.

Se observan gráficamente los resultados para cada filtro sobre la imagen de “Lenna”.

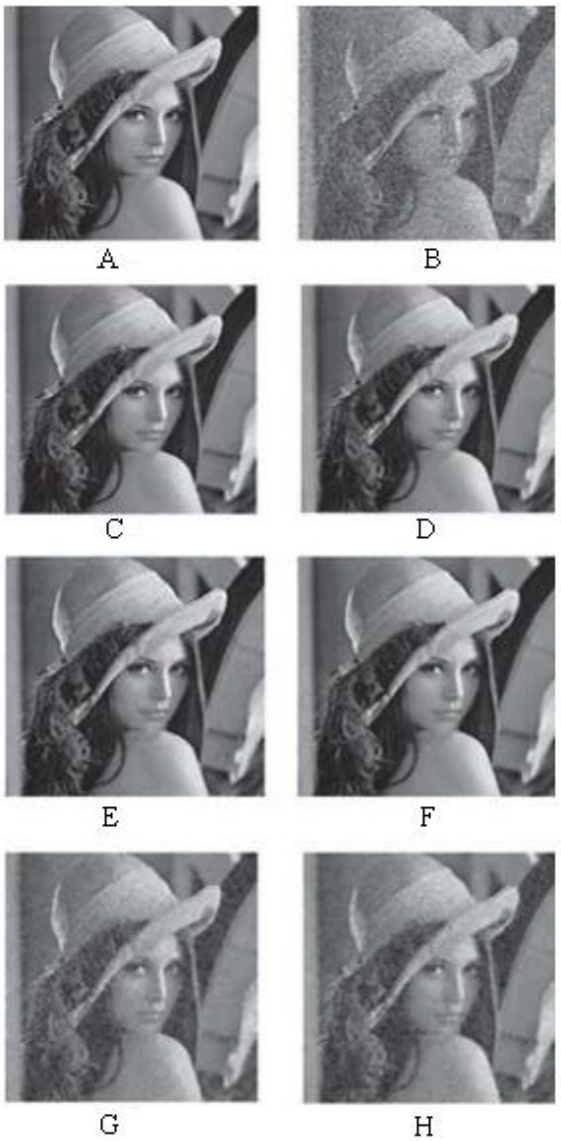


Figura 3-1. Comparación resultados algoritmos.

Tomado de (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

Dónde:

A: Imagen de prueba
B: Imagen ruidosa (40%)
C: Interpolación a través de funciones de base radial
D: Outlier
E: Mediana M
F: Mediana
G: Media M
H: Media

Los resultados, particularmente para el filtro bajo interpolación a través de funciones de base radial, muestra una importante reducción de pixeles ruidosos, sin embargo, el tiempo de ejecución de esta, depende precisamente de la cantidad de pixeles procesados. Ver Figura.

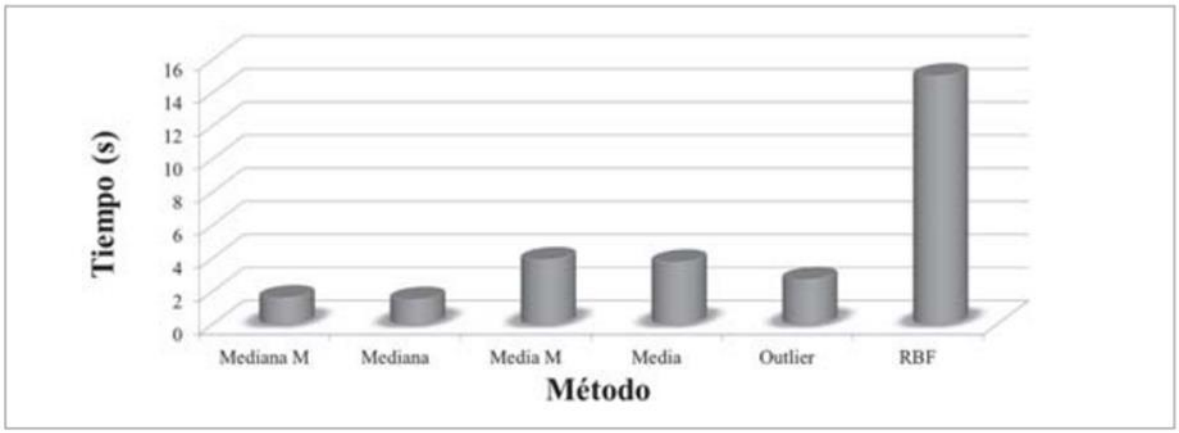


Figura 3-2. Método vs tiempo. Imagen 512 x 512 utilizando Figura 3-1 (A).

Tomado de (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

El filtro de interpolación con funciones de base radiales, toma considerablemente más tiempo, debido a los pixeles procesados y la interpolación calculada. La investigación base, no menciona una tecnología predeterminada para la realización de la comparación, sin embargo, aprovechar el potencial de procesamiento de las tarjetas gráficas, minimizando tiempo de ejecución bajo un algoritmo eficaz, presume un filtro eficaz, eficiente y veloz.

El método consiste básicamente en:

1. Detectar los pixeles ruidosos: Se considera un pixel ruidoso, cuando la diferencia entre su intensidad y la media de las intensidades de la máscara, se encuentra por encima de un umbral alfa definido como la diferencia entre la media y la mediana de la ventana. Se utiliza siempre una máscara de 3x3.
2. Se filtra el vecindario (pixeles cercanos al pixel ruidoso encontrado): Se da la posibilidad que uno o varios de los pixeles que conforman la máscara o vecindario, también presenten ruido, por lo que se debe filtrar para que el nuevo valor no esté afectado por el ruido circundante. Cabe mencionar que, para el presente trabajo, no se tomará en cuenta el filtrado de vecindario.

3. Interpolación del pixel ruido con funciones de base radial: Se debe encontrar el valor que debería tener el pixel, esto a través de funciones de base radial, que encuentran interpoladores dependiendo de la distancia entre el pixel y sus vecinos.

Se hizo una entrega de documentación por parte del asesor temático correspondiente a la investigación. Posteriormente, acontecieron repetidas sesiones explicativas en cuanto a lógica y nivel pseudocódigo de la solución propuesta. Ver la siguiente figura. (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

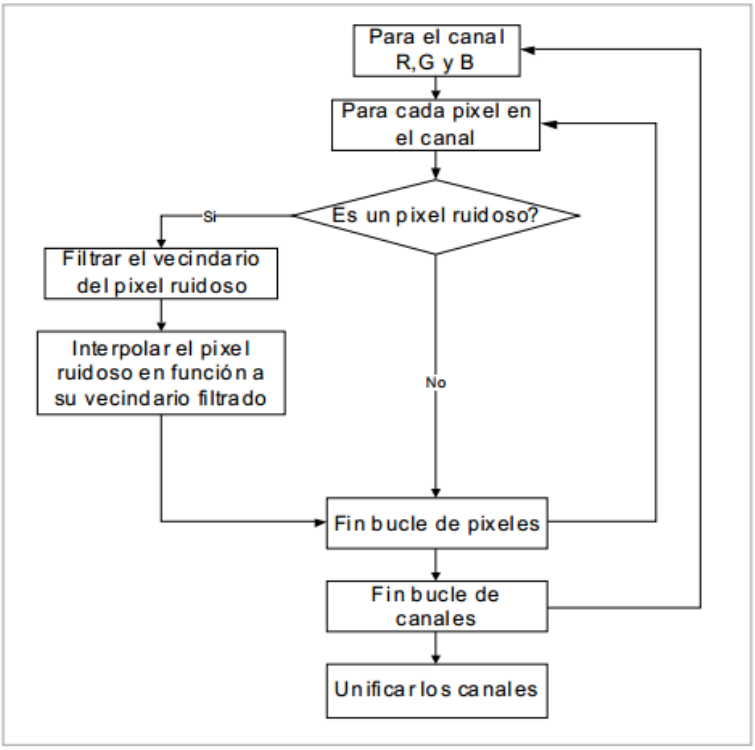


Figura 3-3. Pseudocódigo de la solución.

Tomado de (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

MatLab es un lenguaje de alto nivel, especializado en la manipulación de matrices y fue el lenguaje utilizado por Jaime Echeverry y otros, para implementar el algoritmo de eliminación de ruido impulsivo en la investigación base; se adquirieron las diferentes clases para su posterior análisis, identificando los diferentes métodos, funciones, validaciones entre otras. (Echeverri Arias, Manrique Losada, Moreno, & Bravo, 2009)

La experiencia por parte de los investigadores del presente trabajo, está orientada a tecnologías .NET, específicamente el lenguaje de programación C#, determinar esta forma implementación, o reutilizar métodos o clases, supone un aprendizaje tanto en el lenguaje MatLab, como descarga e instalación de aplicaciones que lo soporten, aumentando el alcance, no es parte integral de la investigación ni tampoco parte de los objetivos.

André R. Brodtkorba, Trond R. Hagena y Martin L. Sætrab brinda información que, evidentemente ayuda al criterio de determinación de la plataforma elegir para la implementación del algoritmo. (Ver Figura 3-9)

Se decide el enfoque particular por NVIDIA CUDA entre otras cosas por la facilidad al desarrollar, porque utiliza C++, (C# deriva de C++, por lo que representa una ventaja para la implementación), existe gran documentación al respecto, existen SDK's disponibles, documentación, y diferentes API's que la implementan.

Ahora bien, con el objetivo de lograr mayor rapidez y aprovechar la experticia de los investigadores del presente trabajo sobre el lenguaje C#, se puede usar un Wrapper como Managedcuda, Cudafy ó Cuda.net, los cuales permiten inyectar código C# y ser traducido automáticamente en lenguaje C/C++ y ser ejecutado en CUDA, ya que el requisito de esta plataforma es el lenguaje base, en este caso C/C++. Información que es detallada en el posterior objetivo de codificación del algoritmo.

En este orden de ideas, los artefactos obtenidos de la investigación de Jaime Echeverri Arias; Bell Manrique Losada; Francisco Javier Moreno; Alejandro Bravo; la explicación del funcionamiento del algoritmo por parte de uno de los investigadores (Jaime Echeverri Arias) , y demás argumentos expuestos en este apartado, llevó a determinar la importancia de implementar el algoritmo en un solo lenguaje de programación y para las dos arquitecturas, partiendo desde cero, sin reutilizar artefactos (clases o métodos en MatLab).

3.1.1. ELEMENTOS DE DISEÑO Y ARQUITECTURA

DIAGRAMA DE CLASES

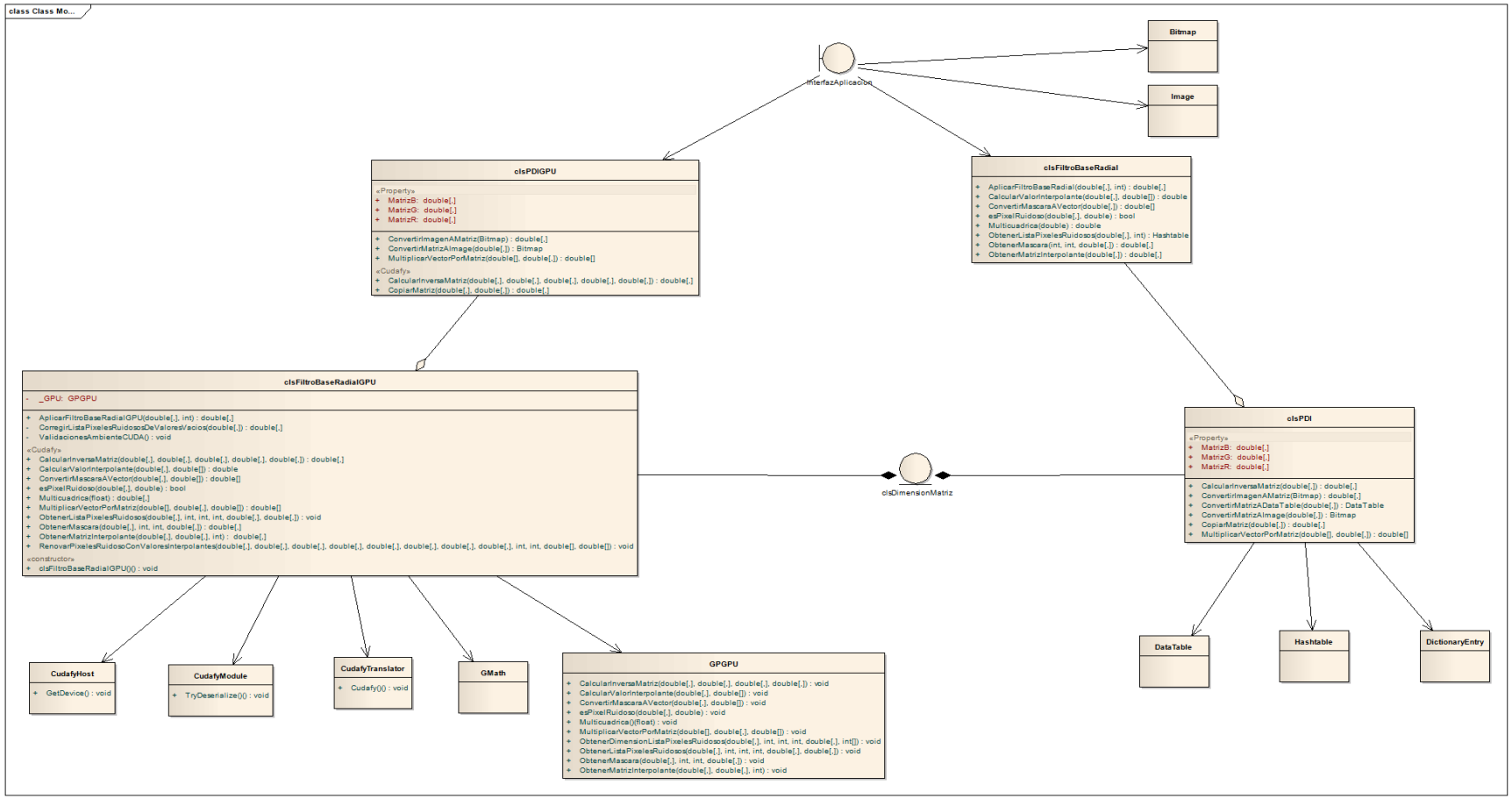


Figura 3-4. Diagrama de clases de la solución.

Se identificaron 4 clases dentro de la librería Cudafy.dll, las cuales permiten la interacción con la GPU, marcando métodos con atributos propios de Cudafy para que sean lanzados y ejecutados sobre la GPU.

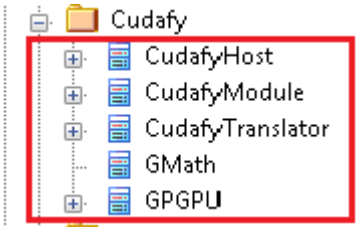


Figura 3-5. Clases asociadas a la librería Cudafy.

De igual manera, se utilizan algunas clases de .NET Framework 4.5

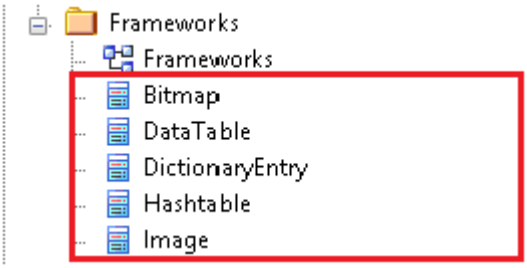


Figura 3-6. Clases utilizadas de .Net Framework 4.5.

La implementación para cada arquitectura, según el modelo de clases, está dada en dos clases,

- **ClsFiltroBaseRadial:** Administra la lógica para la aplicación del algoritmo sobre arquitectura CPU.

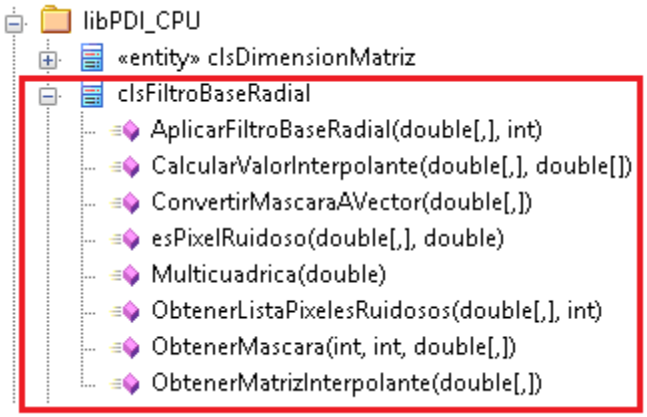


Figura 3-7. Clase que contiene los métodos para aplicar el algoritmo en CPU.

- **ClsFiltroBaseRadialGPU**: Administra la lógica para la aplicación del algoritmo sobre arquitectura GPU.

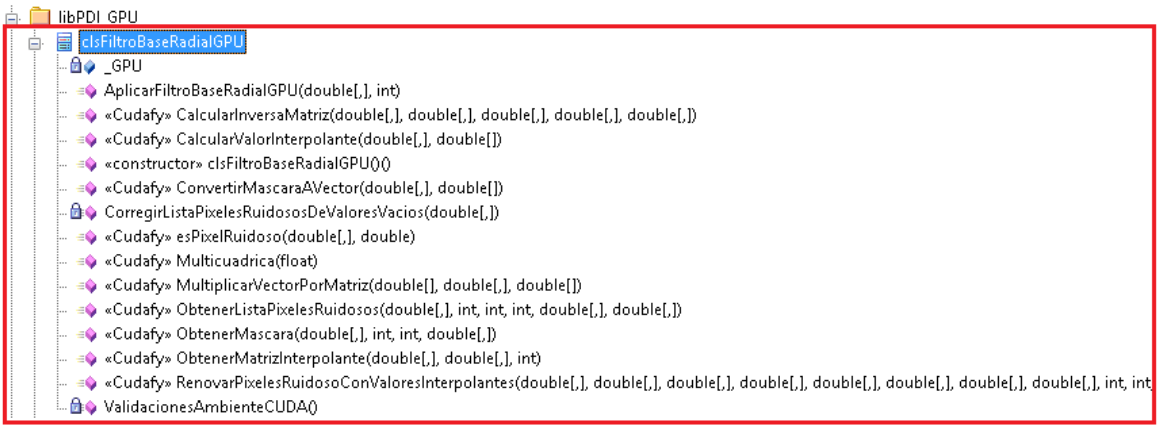


Figura 3-8. Clase que contiene los métodos para aplicar el algoritmo en GPU.

Cada método, cumple una función esencial, se describe a continuación los más relevantes:

AplicarFiltroBaseRadial

Recorre la matriz que contiene información de la imagen, y para cada coordenada verifica si es un pixel ruidoso. En caso de ser pixel ruidoso obtiene la matriz interpolante, la matriz inversa y halla lambda para obtener el valor que debe ser reemplazado en la posición del pixel ruidoso.

ObtenerMascara

Obtiene la máscara o vecindario (se refiere a los pixeles vecinos al pixel evaluado, obteniendo una matriz 3X3) de un pixel.

esPixelRuidoso

Determina si un pixel es ruidoso o no, teniendo en cuenta que el pixel evaluado este en la posición 1,1 de la máscara.

ObtenerMatrizInterpolante

Matriz que contiene el número de comparaciones a realizar por todos los pixeles con el pixel ruidoso.

CalcularValorInterpolante

Se recorre toda la máscara y se obtiene los pesos, estos se suman para dar un valor final que es el valor a reemplazar en el pixel ruidoso.

DIAGRAMAS DE SECUENCIA

La secuencia de acciones para cada clase, se muestra en diagramas de secuencia. Ver anexos 3, 4.

3.2. CODIFICAR EL ALGORITMO PARA SU FUNCIONAMIENTO EN ARQUITECTURAS GPU Y CPU

Se toma como referente documental la investigación realizada por André R. Brodtkorba, Trond R. Hagena y Martin L. Sætrab titulada "Graphics processing unit (GPU) programming strategies and trends in GPU computing" la cual brinda información sumamente importante para elegir sobre cual plataforma (en el caso GPU) es más idónea la implementación del algoritmo. (Brodtkorba, Hagena, & Sætrab, 2013)

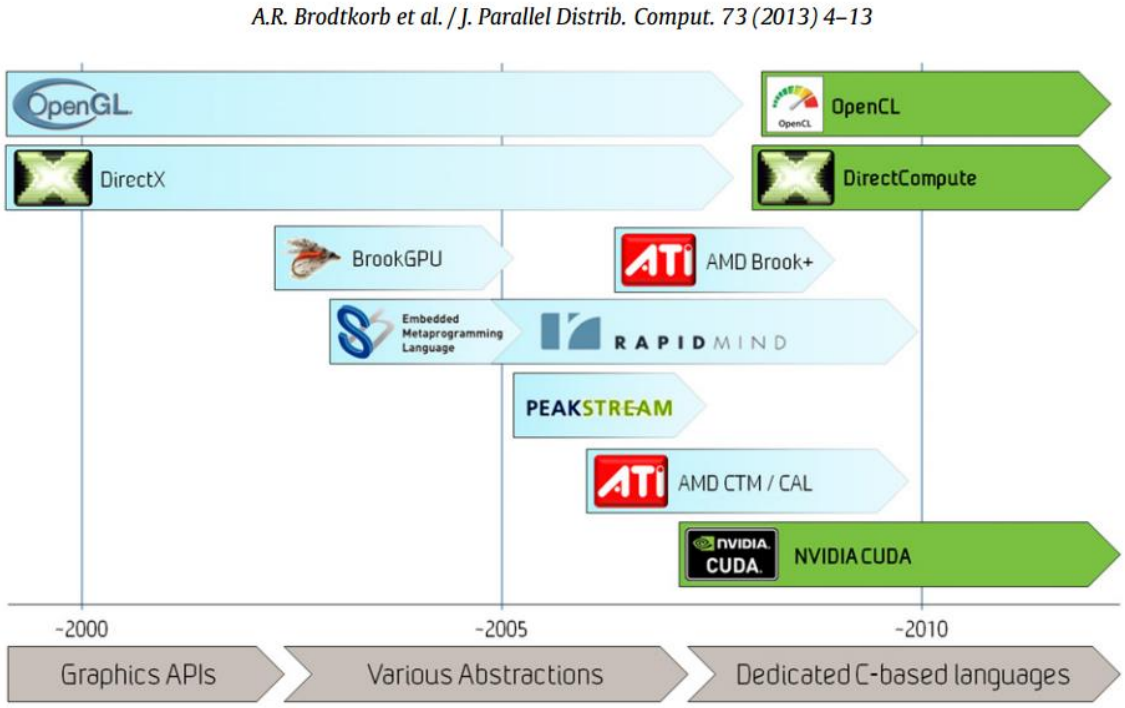


Figura 3-9. Historia de los lenguajes para computación GPU.

Tomado de (Brodtkorba, Hagena, & Sætrab, 2013)

A nivel histórico el surgimiento de GPGPU (GPU con propósito general, es decir mayor aplicabilidad para lo que fue inicialmente concebida) y los lenguajes con mayor abstracción, datan de 2008 con la publicación de AMD CTM/CAL y NVIDIA CUDA.

Se decide el enfoque particular por NVIDIA CUDA por las siguientes razones:

- El enfoque de Nvidia CUDA a nivel de implementación. Facilidad para los desarrolladores, el hecho de utilizar como lenguaje C++ permite un mayor entendimiento para el caso de los desarrolladores con experiencia en otros lenguajes, ya que es sabido, que lenguajes tan populares como Objective C, Java, y C# derivan su estructura de C/C++.

- Evolución Constante. La evolución de esta plataforma va encaminada con el incremento en capacidad del hardware de las tarjetas Nvidia, como ejemplo, las versiones de computación en las que se agregan funcionalidad han mejorado versión a versión hasta actualmente la 5.
- Dimensión de CUDA. Al ser una plataforma, es mucho más robusta que las opciones vigentes DirectCompute y OpenCL, de hecho CUDA ofrece soporte a las mencionadas API's. (NVIDIA Corporation, 2013)
- Curva de aprendizaje. Esto se evidencia con la gran cantidad de SDK's disponibles, documentación, y diferentes API's que la implementan. (NVIDIA Corporation, 2013)
- Algo que es de considerar es el enfoque a tarjetas gráficas Nvidia (sus distintas líneas notebook, desktop, quadro, tesla), esto puede considerarse una ventaja inicialmente ya que obtiene lo mejor del hardware ya que es conocido (fabricante de CUDA y de hardware es el mismo, Nvidia), a diferencia de las demás tarjetas que existen, estas no pueden implementar CUDA. Lo que ofrece OpenCL, por ejemplo, es multiplataforma, es decir, indiferente del hardware GPU, es posible ejecutar código OpenCL.

  Esta característica multiplataforma es un tema muy cuestionado, ya que se conoce en el mercado GPU, los fabricantes tratan de diferenciarse de la competencia ofreciendo ciertas capacidades particulares, haciéndolas en ciertos detalles diferentes al producto de los demás fabricantes.
- A nivel industrial (Educación, investigación, Finanzas computacionales, multimedia entretenimiento, supercomputación, inteligencia y defensa gubernamental) existe una alta aplicabilidad de soluciones que implementan CUDA. (NVIDIA Corporation, 2013), (NVIDIA Corporation, 2012)

Consecuente a la decisión de utilizar CUDA como plataforma base al implementar el algoritmo en GPU, por experiencia laboral de parte de los dos integrantes de la presente implementación, se busca la forma de aplicar tanto el algoritmo en CPU como en GPU, utilizando el framework .NET, usando como lenguaje de programación C#.

Particularmente, para hacer uso del framework .NET y lenguaje C# en GPU, es necesario (para rapidez en implementación) un adaptador (wrapper), que permite el código realizado en lenguaje C# ser traducido automáticamente en lenguaje C/C++ y ser ejecutado en CUDA, ya que el requisito de esta plataforma es el lenguaje base, en este caso C/C++. Para lo anterior se identificaron 3 adaptadores (cuda.net (Harris, 2008), cudafy (Hybrid Computing, 2013), managedcuda

(managedCUDA, 2012)), únicamente cudafy se encuentra en constante mejoramiento y su más reciente versión estable es del mes de mayo de 2013.

La implementación sigue como premisa irrestricta, el desarrollo del algoritmo sin desviarse del pseudocódigo, con el objetivo de construir de la forma más pura, un algoritmo implementado en CPU, para posteriormente implementarse a GPU. Se escoge como tecnología .Net y lenguaje C#, creando una aplicación Windows Forms con una interfaz simple, pero que al mismo tiempo permita identificar y mostrar una imagen con ruido impulsivo (ruido tipo sal y pimienta) antes y después del procesamiento del algoritmo, en ella se carga inicialmente la imagen con características de ruido impulsivo, se define el umbral para evaluar los pixeles y se escoge el tipo de arquitectura con que se desea procesar la imagen. Ver la siguiente Figura.

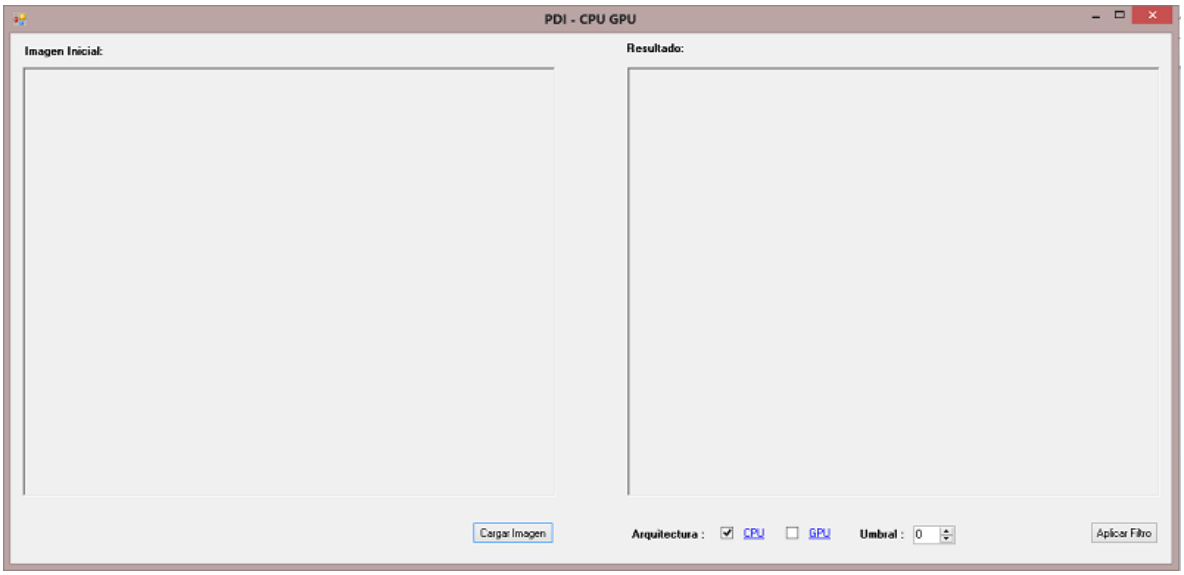


Figura 3-10. Interfaz aplicación pdi_CPU_GPU.

Se inicia la implementación del algoritmo sobre arquitectura CPU, se crean los métodos, funciones y validaciones necesarias.

Ahora bien, el reto manifiesto de esta investigación, se centra en el procesamiento sobre GPU. La complejidad de la implementación radica en ciertas restricciones que actualmente se tienen al utilizar CUDA y por ende el adaptador Cudafy obliga a seguir estas restricciones.

Las siguientes son las más relevantes restricciones identificadas:

- A nivel CUDA, cualquier objeto que no sea tipo primitivo, específicamente double, integer y vectores de cualquier dimensión; no puede ser implementado.

- A nivel CUDA, no es posible inicializar objetos en GPU, si se van a utilizar también en ámbito CPU, es obligatorio, para estos casos, inicializarlo primero en CPU, copiarlas al GPU y posteriormente retornarlas hasta el ámbito CPU (relacionado y detallado en el flujo de proceso CPU-GPU).
- A nivel Cudafy, al crear un vector de cualquier dimensión, está inicializado con valores Nan (valores no numéricos).
- A nivel Cudafy, todos los métodos que hagan parte de la solución que se ejecute, deben ser públicos y estáticos, a excepción del primer método, el cual inicializa Cudafy.

Es posible en futuras versiones, algunas de estas restricciones pierdan vigencia, pero hasta el momento versión CUDA 5, se mantienen las enunciadas.

A través de la documentación propia del grupo desarrollo de tecnología de Nvidia (Developer Technology Group (Woolley, 2012) se presenta el flujo de proceso entre CPU-GPU

1. Se copia la información de la memoria CPU a la memoria GPU

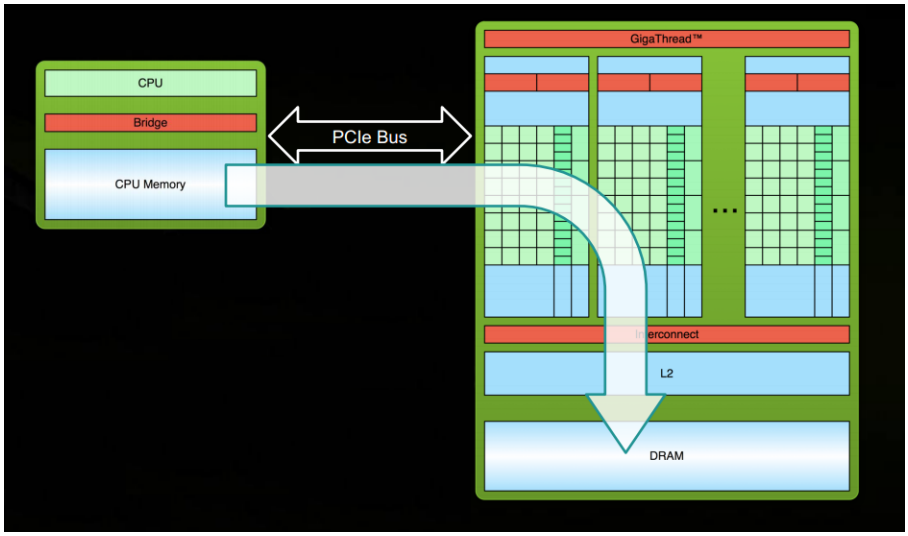


Figura 3-11. Flujo procesamiento CPU a GPU.

Figura tomada y modificada de (Woolley, 2012)

2. Se carga el programa GPU y se ejecuta, cacheando la información sobre el chip para desempeño

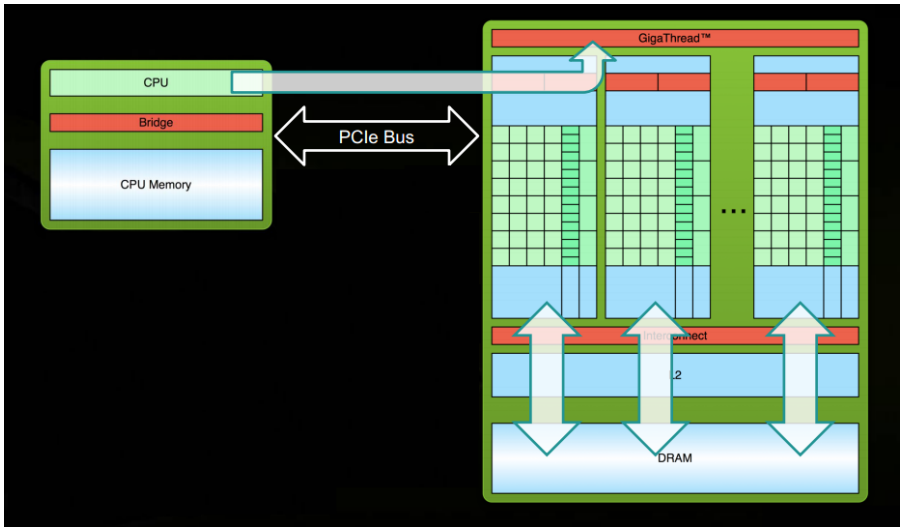


Figura 3-12. Flujo procesamiento CPU a GPU, Carga y Ejecución.

Figura tomada y modificada de (Woolley, 2012)

3. Copia los resultados de la memoria GPU a la memoria CPU.

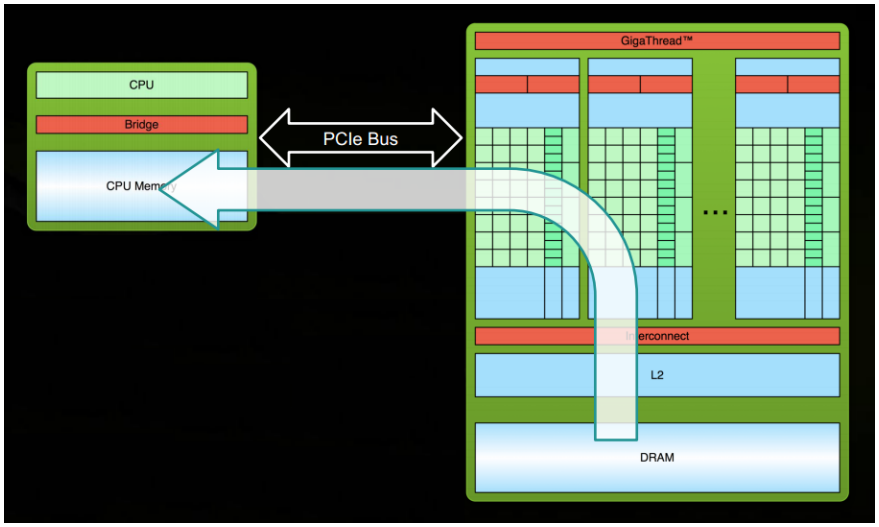


Figura 3-13. Flujo procesamiento GPU a CPU.

Figura tomada y modificada de (Woolley, 2012)

3.3. ELABORAR PRUEBAS SOBRE EL ALGORITMO IMPLEMENTADO.

Con la implementación terminada, se debe ejecutar un set de pruebas que permita asegurar la calidad de la aplicación, encontrar errores, falencias y posteriormente identificar tiempos de respuesta, precisión en las dos arquitecturas.

Inicialmente al ejecutar las pruebas en GPU se identificaron ciertos problemas:

- El compilador C++ de visual studio es utilizado por Cudafy: al iniciar pruebas no era posible ejecutar ningún proceso de GPU, esto se debe a que Cudafy al tomar el código en lenguaje C#, lo transforma a un .ptx para ser ejecutado por el compilador C++, al no tener la ubicación del compilador. Es por esto que se debe agregar en la variable de sistema Path la ubicación del compilador C++ (los pasos son detallados en el manual de instalación). Ejemplo: {archivos de programa}\{versión VS}\VC\bin.
- El dispositivo GPU tiene un tiempo limitado de procesamiento: cuando se utiliza como dispositivo de procesamiento la tarjeta primaria del equipo, esta tiene un parámetro dado por el sistema operativo, llamado TDR (Timeout Detection and Recover of GPU's), esto tiene que ver con recuperar en un determinado tiempo, (minutos, valor numérico a dar en el TDR) la tarjeta gráfica evitando que en alguna operación gráfica haya ocasionado el efecto de congelamiento en pantalla y el usuario piense que se ha bloqueado el equipo. El TDR con un valor bajo es un problema para la aplicación implementada, ya que si se realizan pruebas con imágenes de grandes dimensiones sucede una excepción por parte de Cudafy donde no permite finalizar, es por esto que se debe ampliar el TDR. Estas modificaciones deben realizarse en el registro de Windows (los pasos son detallados en el manual de instalación)

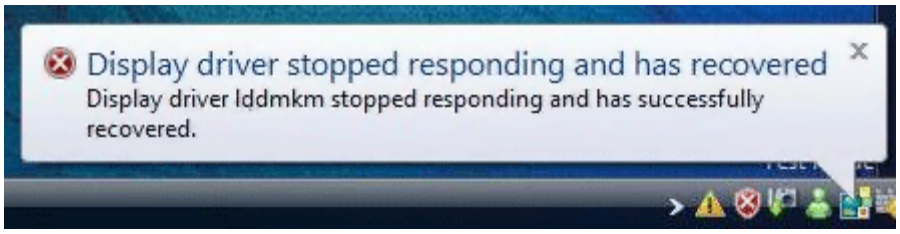


Figura 3-14. Imagen error TDR.

* Imagen del error al momento de ejecutar imágenes de grandes dimensiones

Las siguientes fueron las acciones realizadas para llevar a cabo el set de pruebas:

- Se generan alrededor de 12 imágenes en blanco y negro, desde dimensiones pequeñas (10X10) hasta imágenes grandes (4400X2900).

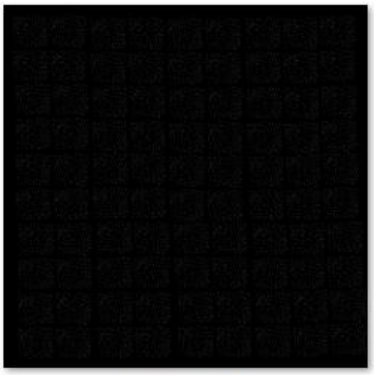
Imagen 1	**Imagen 2**	**Imagen 3**
 ImagenRuido1.jpg	 ImagenRuido2.png	 ImagenRuido3.png
Dimensiones:10*10	Dimensiones:1000*1000	Dimensiones:320*480
Imagen 4	**Imagen 5**	**Imagen 6**
 ImagenRuido4.jpeg	 ImagenRuido5.jpg	 ImagenRuido6.jpg
Dimensiones:1680*1050	Dimensiones: 1600*900	Dimensiones: 1000*774

Tabla 3-1. Conjunto 1 Imágenes de Prueba.

Dimensiones: ancho por alto dado en pixeles.

Imagen 7	Imagen 8	Imagen 9
 ImagenRuido7.jpg	 ImagenRuido8.jpg	 ImagenRuido9.jpg
Dimensiones:400*300	Dimensiones:640*480	Dimensiones:281*279
Imagen 10	**Imagen 11**	**Imagen 12**
 ImagenRuido10.jpg	 ImagenRuido11.jpg	 ImagenRuido12.jpg
Dimensiones:1801*1200	Dimensiones:1600*985	Dimensiones:4408*2904

Tabla 3-2. Conjunto 2 Imágenes de Prueba.

Dimensiones: ancho por alto dado en pixeles.

- Para las pruebas, en todos los casos se aplica el mismo Umbral 80, esto quiere decir, que la aplicación considerará como ruido cualquier valor que sea igual o superior a 80 en su intensidad de color, en este caso, cabe recordar, que el valor 0 es color negro y el valor 255 es blanco.

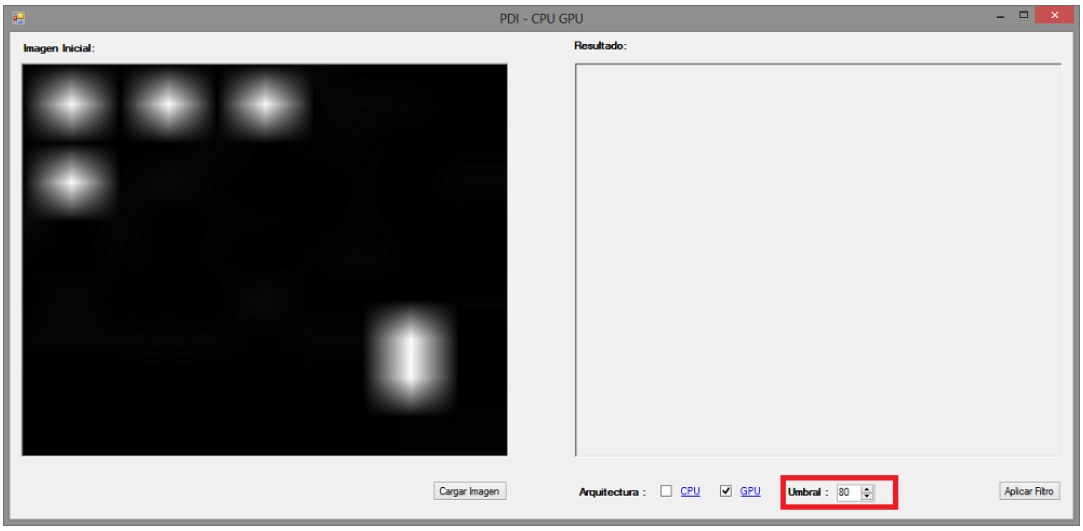
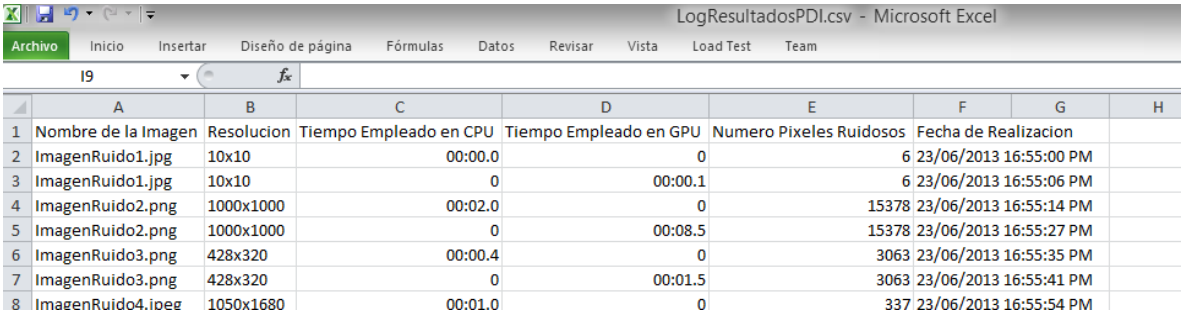


Figura 3-15. Interfaz aplicación pdi_CPU_GPU modificación Umbral.

- En la aplicación se implementó una librería que permite establecer el tiempo desde el inicio hasta el final de ejecución del método principal de aplicación del filtro, ya sea CPU o GPU. La librería registra el tiempo en un archivo de tipo .cvs, esto permite utilizar la información para tabularla y generar gráficos que permitan dar mayor detalle para interpretación. Se genera un archivo LogResultadosPDI.csv en la carpeta de documentos (mis documentos) y se detalla: Nombre de la imagen, Resolución, Tiempo empleado en CPU, Tiempo empleado en GPU, Número Pixeles Ruidosos y la fecha de realización.



	A	B	C	D	E	F	G	H
1	Nombre de la Imagen	Resolucion	Tiempo Empleado en CPU	Tiempo Empleado en GPU	Numero Pixeles Ruidosos	Fecha de Realizacion		
2	ImagenRuido1.jpg	10x10	00:00.0	0	6	23/06/2013 16:55:00 PM		
3	ImagenRuido1.jpg	10x10	0	00:00.1	6	23/06/2013 16:55:06 PM		
4	ImagenRuido2.png	1000x1000	00:02.0	0	15378	23/06/2013 16:55:14 PM		
5	ImagenRuido2.png	1000x1000	0	00:08.5	15378	23/06/2013 16:55:27 PM		
6	ImagenRuido3.png	428x320	00:00.4	0	3063	23/06/2013 16:55:35 PM		
7	ImagenRuido3.png	428x320	0	00:01.5	3063	23/06/2013 16:55:41 PM		
8	ImagenRuido4.jpeg	1050x1680	00:01.0	0	337	23/06/2013 16:55:54 PM		

Figura 3-16. Vista información archivo LogResultadosPDI.csv.

- Para la ejecución de las pruebas, se planteó la siguiente estrategia
    - Se utilizan 2 equipos portátiles con características similares:

Componentes	Equipo 1	Equipo 2
Procesador	Interl Core i7-3610QM	Interl Core i7-3610QM
	Velocidad: 2.30GHz Núcleos Físicos: 4 Procesadores Lógicos: 8	Velocidad: 2.30GHz Núcleos Físicos: 4 Procesadores Lógicos: 8
Tarjeta Gráfica	GeForce GT 640M LE Capability: 2.1	GeForce GT 630 Capability: 2.1
	Memoria: 2GB	Memoria: 2GB
RAM	Memoria: 8GB	Memoria: 8GB

Tabla 3-3. Componentes hardware equipos de prueba.

    - Cargar cada una de las 12 imágenes, por cada imagen se aplica el filtro en CPU y GPU
    - Los datos recopilados en el archivo "logResultadosPDI.csv" son tabulados en un nuevo documento "ResultadosPDI.xslx".
    - En el documento "ResultadosPDI.xslx", se realizan gráficos para la interpretación de los resultados.

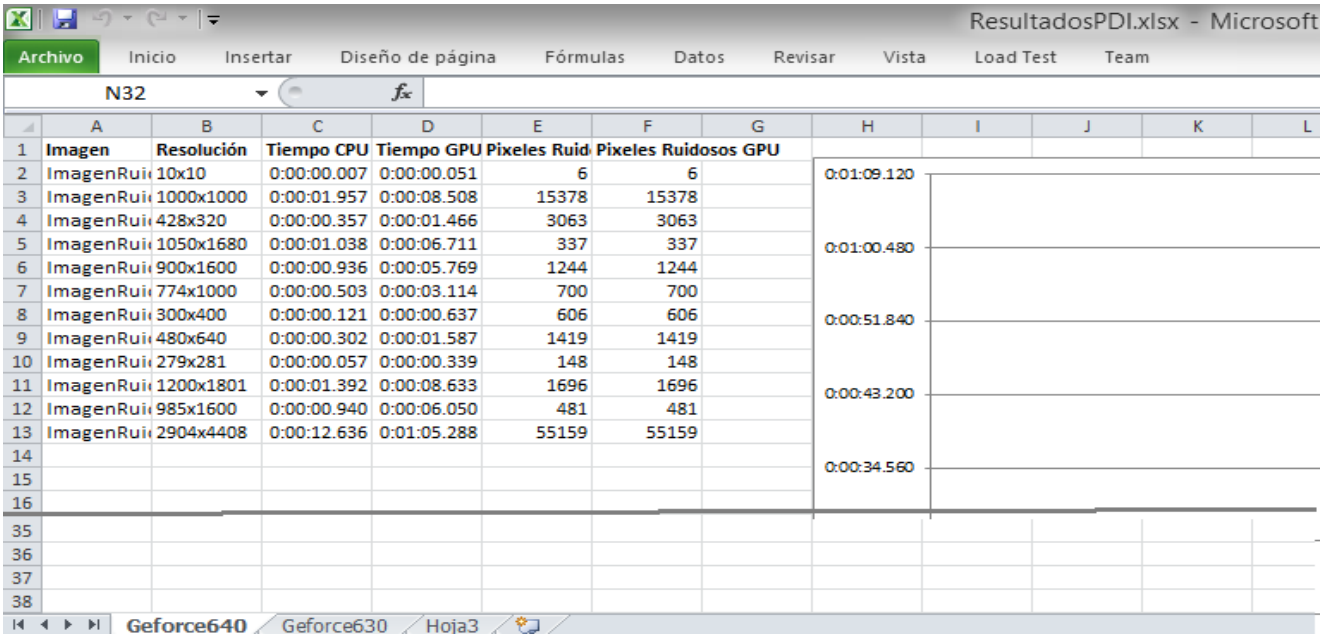


Figura 3-17. Vista información tabulada en ResultadosPDI.xslx.

3.4. ESTABLECER COMPARATIVOS EN TÉRMINOS DE TIEMPOS DE RESPUESTA Y PROCESAMIENTO DEL ALGORITMO BAJO AMBAS ARQUITECTURAS (GPU Y CPU)

En el proceso de construcción de la aplicación se establecieron diferentes detalles para auditar y de esta manera generar información de análisis.

La aplicación cada vez que se aplica el filtro permite registrar el tamaño de la imagen, el número de pixeles ruidosos que fueron procesados y sobre qué arquitectura se ejecutó.

Al ejecutar la estrategia de pruebas mencionada anteriormente, esta información es ordenada y consolidada en un único documento, esto para generar gráficos y valores tabulados que permiten una rápida comprensión de los resultados.

Las tablas con los resultados, tienen las siguientes columnas

Imagen: Nombre del archivo

Resolución: dimensión en ancho por alto de la imagen

Tiempo CPU: Tiempo empleado por la CPU para procesar la imagen, expresado en horas, minutos, segundos y milisegundos

Tiempo GPU: Tiempo empleado por la GPU para procesar la imagen, expresado en horas, minutos, segundos y milisegundos

Pixeles ruidosos CPU: cantidad de pixeles identificados por el algoritmo ejecutado en CPU

Pixeles ruidosos GPU: cantidad de pixeles identificados por el algoritmo ejecutado en GPU

Los resultados generados en el equipo 1

Imagen	Resolución	Tiempo CPU	Tiempo GPU	Pixeles Ruidosos CPU	Pixeles Ruidosos GPU
ImagenRuido1.jpg	10x10	0:00:00.007	0:00:00.051	6	6
ImagenRuido2.png	1000x1000	0:00:01.957	0:00:08.508	15378	15378
ImagenRuido3.png	428x320	0:00:00.357	0:00:01.466	3063	3063
ImagenRuido4.jpeg	1050x1680	0:00:01.038	0:00:06.711	337	337
ImagenRuido5.jpg	900x1600	0:00:00.936	0:00:05.769	1244	1244
ImagenRuido6.jpg	774x1000	0:00:00.503	0:00:03.114	700	700
ImagenRuido7.jpg	300x400	0:00:00.121	0:00:00.637	606	606
ImagenRuido8.jpg	480x640	0:00:00.302	0:00:01.587	1419	1419
ImagenRuido9.jpg	279x281	0:00:00.057	0:00:00.339	148	148
ImagenRuido10.jpg	1200x1801	0:00:01.392	0:00:08.633	1696	1696
ImagenRuido11.jpg	985x1600	0:00:00.940	0:00:06.050	481	481
ImagenRuido12.jpg	2904x4408	0:00:12.636	0:01:05.288	55159	55159

Tabla 3-4. Resultados pruebas en equipo 1.

Gráfico de resultados Equipo 1 – Tiempo CPU vs. Tiempo GPU

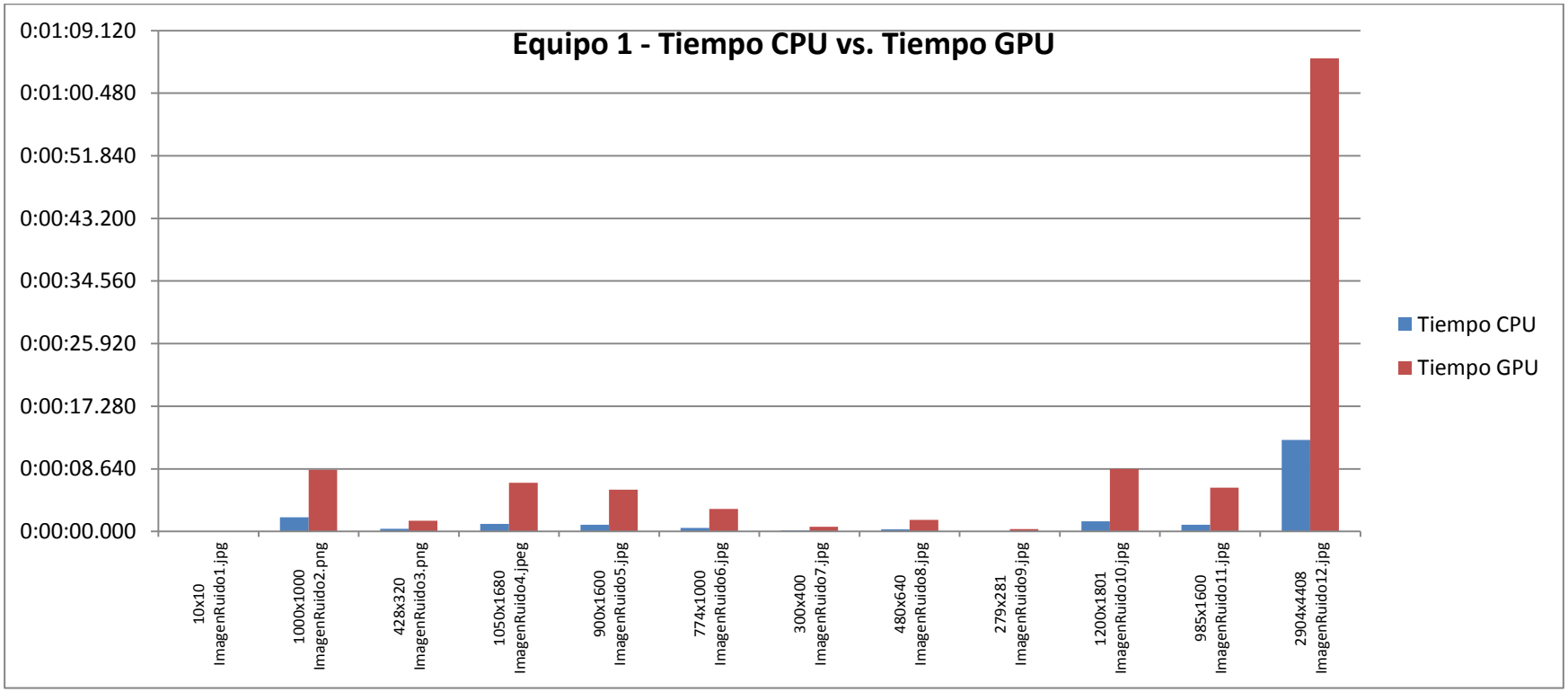


Figura 3-18. Resultados Equipo 1, Tiempo CPU versus Tiempo GPU.

Los resultados generados en el equipo 2

Imagen	Resolución	Tiempo CPU	Tiempo GPU	Pixeles Ruidosos CPU	Pixeles Ruidosos GPU
ImagenRuido1.jpg	10x10	0:00:00.000	0:00:00.030	6	6
ImagenRuido2.png	1000x1000	0:00:01.370	0:00:08.130	15378	15378
ImagenRuido3.png	428x320	0:00:00.250	0:00:01.500	3063	3063
ImagenRuido4.jpeg	1050x1680	0:00:00.720	0:00:06.330	337	337
ImagenRuido5.jpg	900x1600	0:00:00.650	0:00:05.440	1244	1244
ImagenRuido6.jpg	774x1000	0:00:00.350	0:00:03.090	700	700
ImagenRuido7.jpg	300x400	0:00:00.080	0:00:00.600	606	606
ImagenRuido8.jpg	480x640	0:00:00.210	0:00:01.490	1419	1419
ImagenRuido9.jpg	279x281	0:00:00.040	0:00:00.320	148	148
ImagenRuido10.jpg	1200x1801	0:00:00.970	0:00:08.300	1696	1696
ImagenRuido11.jpg	985x1600	0:00:00.650	0:00:05.710	481	481
ImagenRuido12.jpg	2904x4408	0:00:08.730	0:01:01.790	55159	55159

Tabla 3-5. Resultados pruebas en equipo 2

Gráfico de resultados Equipo 2 – Tiempo CPU vs. Tiempo GPU

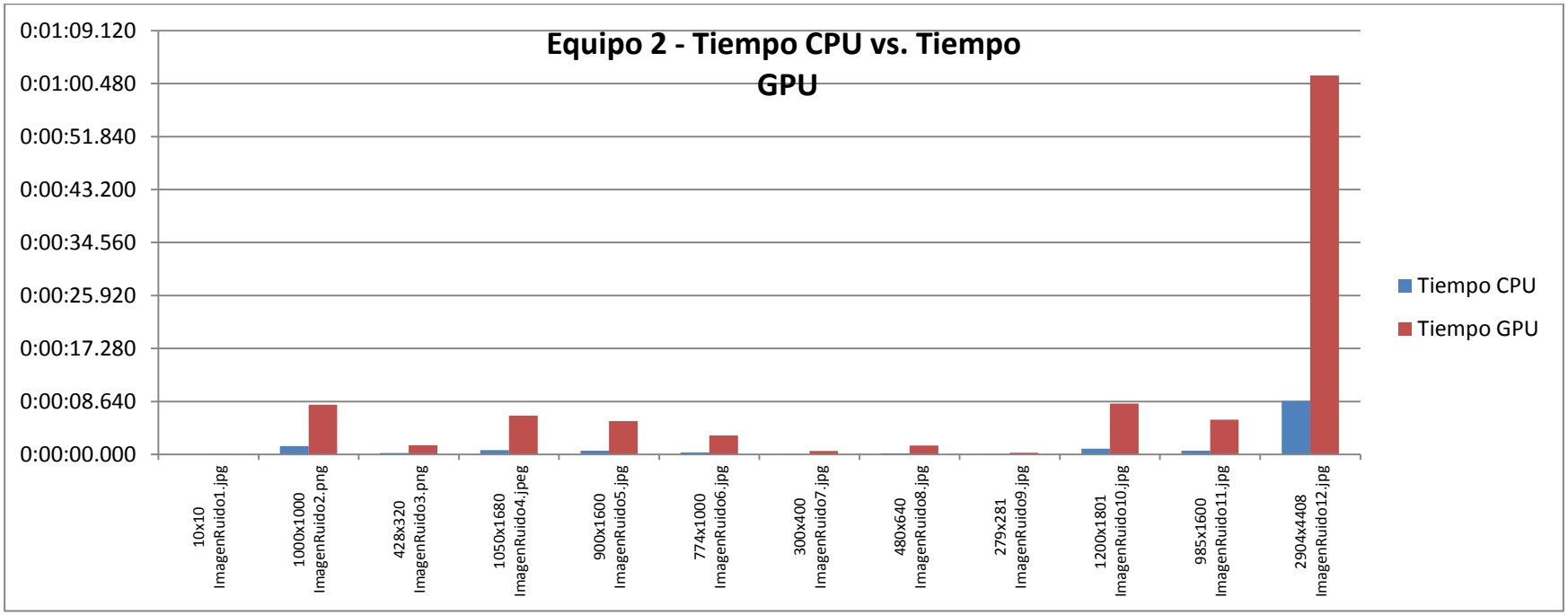


Figura 3-19. Resultados Equipo 2, Tiempo CPU versus Tiempo GPU.

ANEXOS

Anexo 1. Inventario de capacidades de investigación

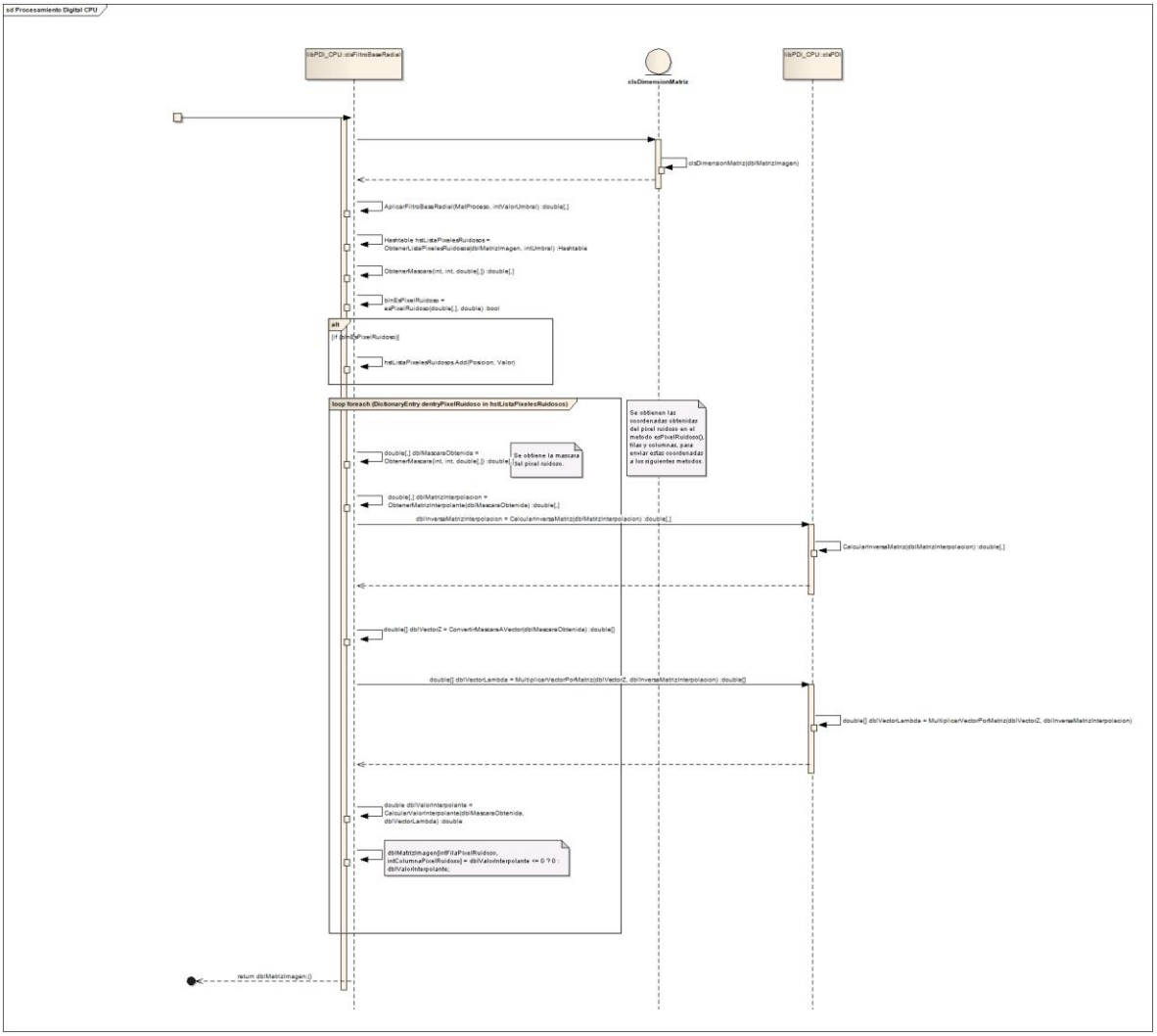
Grupo de Investigación							
Capital Humano		Nombre completo	Títulos De Pregrado	Títulos De Posgrado	Experiencia En Docencia Nivel de Pregrado	Experiencia En Docencia Nivel de Posgrado	Experiencia Profesional relevante (cargo, empresa y período)
	Líder	Neilor Esteban Diaz López	1. Técnico profesional en Ingeniería de Sistemas. 2. Ingeniería de Sistemas				Ingeniero de Desarrollo, Intergrupo S.A., Inicio: 14/03/2011
	Investigadores (as)	1. Sergio Alberto Cano Ocampo 2. Neilor Esteban Diaz López	1.Tecnología de Sistemas de Información, Ingeniería en Sistemas de información 2. Ingeniería en Sistemas de Información				1. Ingeniero de Desarrollo, Intergrupo S.A., Inicio: 15/06/2010 2. Ingeniero de Desarrollo, Intergrupo S.A., Inicio: 14/03/2011
	Técnicos y auxiliares de laboratorio						

Proyectos de investigación	Título	Duración prevista o período de ejecución	Costo (o presupuesto aprobado)	Otros grupos o instituciones participantes
En ejecución	Implementación de un algoritmo para eliminación de ruido impulsivo en imágenes y análisis comparativo de tiempos de respuesta bajo arquitectura GPU y CPU	28 días.	$ 30.420.000	Universidad de Medellín

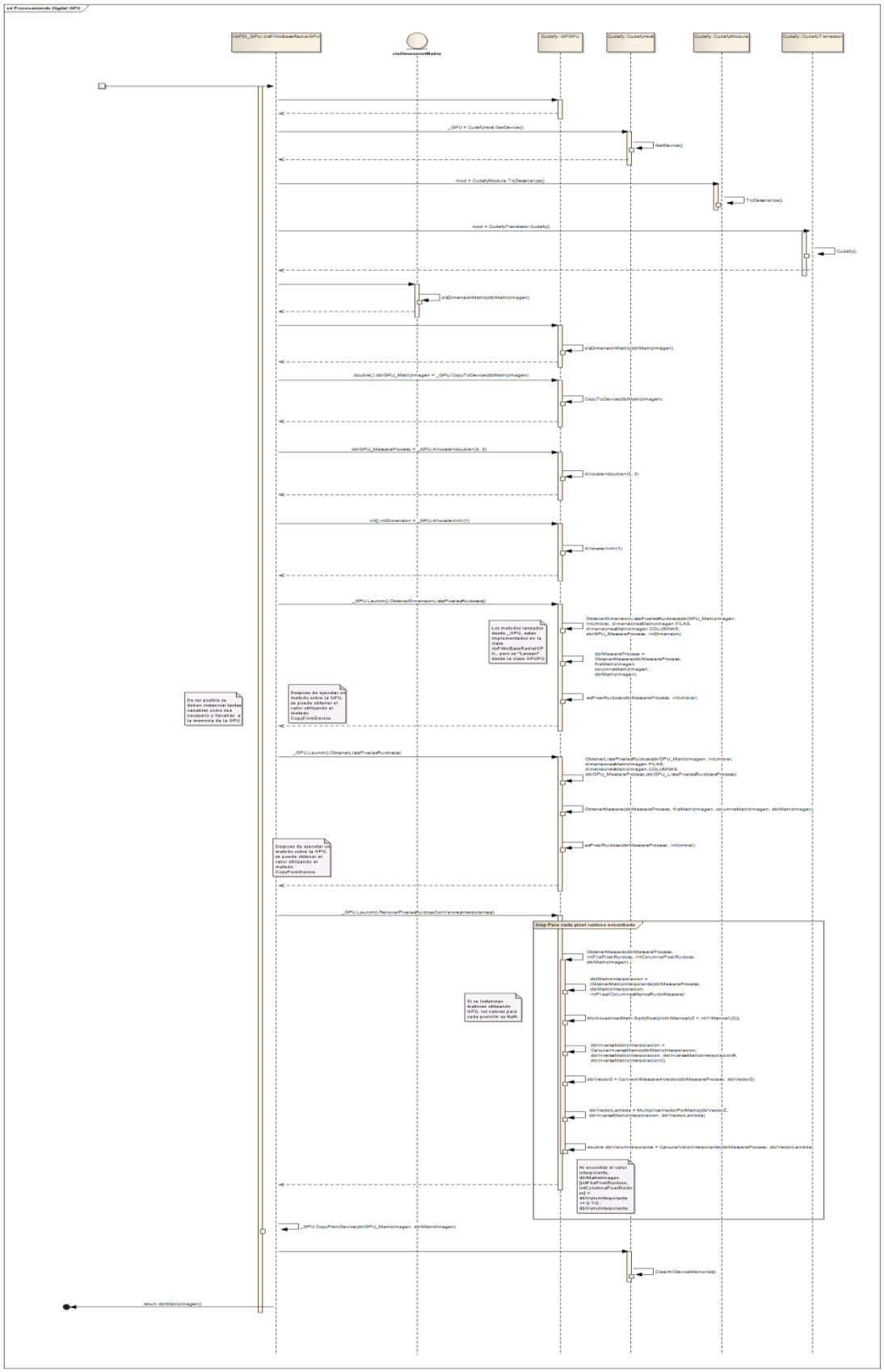
Anexo 2. Cronograma de Actividades

No	DESCRIPCIÓN DE LA ACTIVIDAD	2012 (meses)					
		1	2	3	4	5	6
1.1	Consultar información que exista en las diferentes bases de datos de investigación.	X					
1.2	Recopilar, analizar y extraer la información relevante para el proyecto.	X					
2.1	Adquirir los elementos necesarios para la codificación, llámese equipos de cómputo con tarjetas gráficas especializadas (GPU).	X					
2.2	Diseñar y generar la interfaz de interacción.		X				
2.3	Diseñar y Codificar el algoritmo en base radial para el tratamiento de imágenes.			X	X		
3.1	Definir un plan de pruebas con el fin de asegurar la calidad y funcionamiento de los algoritmos.					X	
4.1	Obtener los tiempos de respuesta de cada algoritmo bajo las dos arquitecturas propuestas.					X	
4.2	Medir y comparar los datos obtenidos						X

Anexo 3. Diagrama de secuencia ProcesamientoDigitalCPU



Anexo 4. Diagrama de secuencia ProcesamientoDigitalGPU



GLOSARIO

API: (Application Programming Interface) Interfaz de programación de aplicaciones, conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otros software como una capa de abstracción.

CMMI: (Capability Maturity Model Integration) modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.

CPU: (Central Processing Unit) también llamado procesador, componente principal del ordenador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos

CUDA: (Compute Unified Device Architecture) es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

FILTRO: en informática, es un algoritmo o solución codificada que sirve para procesar un conjunto de datos.

FRAMEWORK: (marco de trabajo) infraestructura digital, puede servir de base para la organización y desarrollo de software.

GPGPU: (General-Purpose Computing on Graphics Processing Units) utilización de una unidad gráfica de procesamiento (GPU), donde generalmente se utiliza para computación gráfica, a utilizarla en aplicación que normalmente son manipuladas por la CPU (Unidad Central de Procesamiento).

GPU: (Graphics Processing Unit) coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo de la CPU.

HARDWARE: partes tangibles de un sistema informático. Sus componentes son: eléctricos, electrónicos, electromecánicos y mecánicos.

INTELIGENCIA ARTIFICIAL: rama de la ciencia computacional y tecnología que estudia y desarrolla máquinas y software inteligente. Donde inteligencia es la forma de percibir su ambiente y tomas acciones que maximicen las oportunidades de éxito.

PIXEL: es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea esta una fotografía, un fotograma de vídeo o un gráfico.

PSEUDOCÓDIGO: una descripción informal de alto nivel de un algoritmo informático de programación, compacto e informal, que utiliza las convenciones estructurales de un lenguaje de programación verdadero , pero que está diseñado para la lectura humana en lugar de la lectura mediante máquina, y con independencia de cualquier otro lenguaje de programación.

RUIDO IMPULSIVO: conocido como ruido sal y pimienta los píxeles de la imagen son muy diferentes en color o intensidad a los píxeles circundantes. El hecho que define este tipo de ruido es que el pixel ruidoso en cuestión no tiene relación alguna con los píxeles circundantes.

SDK: (Software Development Kit) conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto.

SOFTWARE: equipamiento lógico o soporte lógico de un sistema informático, comprende el conjunto de componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos que son llamados hardware.

UMBRAL: valor que permite definir diferencias en un conjunto de datos.

VISIÓN ARTIFICIAL: subcampo de la inteligencia artificial, incluye métodos para procesar, analizar, entender imágenes en general, información de alta dimensión desde el mundo real para producir información numérica o simbólica.

WRAPPER: Es una capa, porción de código, elementos que encapsulan la lógica interna de un proceso o tarea final.
Existen dos tipos:
Wrappers de abstracción: oculta/abstraen la tareas final para que sea lo último en enlazar.
Wrappers controladores/motor: utilizados para agrega un contexto lógico o una lógica específica que no (o se supone que no) puede ser realizada por la tarea final.

CONCLUSIONES

- A través de las pruebas realizadas y de acuerdo a la forma de implementación del algoritmo para las dos arquitecturas, es posible determinar los tiempos de respuesta.
  Sobre estos resultados de las pruebas se determina un tiempo de respuesta más alto en la arquitectura GPU respecto a la arquitectura CPU. En todos los casos de prueba se evidencia esta diferencia.

- Existen diversas formas de aprovechar el potencial de procesamiento de los dispositivos GPU, entre ellos una forma alternativa es emplearla como arquitectura de apoyo para la CPU. Es decir, es posible crear soluciones donde parcialmente sea utilizada la arquitectura GPU, manteniendo como principal ámbito la CPU.

- Las limitaciones en cuanto a objetos que pueden se crear en el ámbito GPU, hacen necesaria la reestructuración del código, ya que inicialmente se pensaba en reutilización del algoritmo implementado en CPU, pero determinadas restricciones sobre los dispositivos GPU no lo permiten.
  Este implementación adaptada a los dispositivos GPU exige mayor tiempo en el diseño del algoritmo ya que se deben buscar alternativas al uso de las facilidades del framework .net, las cuales no están disponibles, lo cual desencadenaría en una implementación poco optimizada de la solución.

- El hecho de aplicar un set de pruebas con resultados en tiempo de respuesta alto para arquitectura GPU, no hacen de esta implementación una solución irrelevante, ya que como se especificó, en el detalle de las pruebas se utilizaron tarjetas gráficas de portátiles, los resultados en tarjetas gráficas profesionales pueden dar resultados mucho más bajos en cuanto a tiempo de respuesta, lo cual permite que la investigación pueda ser continuada y abordada para otras aplicaciones.

- El wrapper cudafy permite una forma ágil de implementación para dispositivos GPU, su constante evolución en mejoramiento de versiones lo perfila como una herramienta a aplicar en futuros proyectos en los cuales se incluya la computación con cualquier tipo de dispositivo que utilice gráficos (móvil, tablet, pc).

- Los dispositivos GPU se presentan como una alternativa a considerar a la hora de requerir procesamiento adicional o intensivo sobre una necesidad en particular, la constante evolución sobre la capacidad de hardware sumado a la evolución en cuanto a las herramientas de programación en ellas, facilitan la implementación y ayudan al incremento en el uso de estas alternativas.

- Para futuras investigaciones es importante considerar hardware de procesamiento profesional, ya que tanto la arquitectura CPU como la arquitectura GPU tienen líneas de productos enfocadas para alto performance.

- La implementación por medio del wrapper cudafy permite ser utilizada en otras plataformas al realizar pequeñas adaptaciones a la implementación realizada. Esto supone una gran ventaja ya que abre la posibilidad de experimentar el desempeño en diferente dispositivos que no tengan la tecnología CUDA (dispositivo que no sean producidos por NVidia) añadiendo a esta solución una característica especial, aplicable en múltiples dispositivos hardware al realizar pequeñas modificaciones al dispositivo destino.

BIBLIOGRAFÍA

managedCUDA. (2012, Diciembre 11). Recuperado el Junio 8, 2013, de http://managedcuda.codeplex.com/

Association for Computing Machinery. (2008). Queue - GPU Computing. *ACM Queue*, 37.

Boncelet, C. (2009). *Chapter 7 - Image Noise Models.* Recuperado el Agosto 4, 2013, de The Essential Guide to Image Processing (Second Edition): http://dx.doi.org/10.1016/B978-0-12-374457-9.00007-X

Brodtkorba, A. R., Hagena, T. R., & Sætrab, M. L. (2013, Enero). *Graphics processing unit (GPU) programming strategies and trends in GPU computing.* Recuperado el Junio 8, 2013, de http://dx.doi.org/10.1016/j.jpdc.2012.04.003

Brodtkorba, A., Kim, C., Chhugani, J., Deisher, M., Kim†, D., Nguyen, A., y otros. (2010). Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. *ACM New York*, 451-460.

Bui, P., & Brockman, J. (2009). *Performance Analysis of Accelerated Image Registration Using GPGPU.* Recuperado el Junio 07, 2013, de http://dl.acm.org/citation.cfm?id=1513900

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., & Skadron, K. (2008, Octubre 10). *A performance study of general-purpose applications on graphics processors using CUDA.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=1412827

Echeverri Arias, J. A., Manrique Losada, B., Moreno, F. J., & Bravo, A. (2009, Julio). *Images improvement using radial basis functions.* Recuperado el Junio 08, 2013, de http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-33242009000300003

Farber, R. (2011). *CUDA Application Design and Development.* Obtenido de http://books.google.com.co/books?id=Y-XmJO2uwvMC&printsec=frontcover&hl=es#v=onepage&q&f=false

Fatahalian, K., & Houston, M. (2008). GPUs: A Closer Look. *ACM Queue*.

Fowers, J., Brown, G., Cooke, P., & Stitt, G. (2012). *A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=2145704

García Elías, H. A., & Ramírez Cruz, J. F. (2003). *Método Basado en Redes Neuronales Wavelet para Eliminar Ruido en Espectros Estelares.* Presentado en el II Congreso Español de Informática.

Gómez Moreno, H., Maldonado Bascón, S., Utrilla Manso, M., & Martín Martín, P. (s.f.). Eliminación de ruido impulsivo en imágenes mediante el uso de máquinas de vectores soporte. *Universidad de Alcalá*.

Harris, M. (2008, Julio 10). *CUDA.NET.* Recuperado el Junio 8, 2013, de http://gpgpu.org/2008/07/10/cudanet

Hybrid Computing. (2013, Mayo 8). *CUDAfy.NET.* Recuperado el Junio 8, 2013, de http://cudafy.codeplex.com/

Igual, F. D., Mayo, R., Hartley, T. D., Çatalyürek, Ü. V., Ruiz, A., & Ujaldon, M. (2011, Noviembre). *Color and texture analysis using emerging parallel architectures.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=2076556.2076568

Krewell, K. (2009, Diciembre 16). *DIFFERENCE CPU AND A GPU.* Recuperado el 08 12, 2013, de http://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

Kumar, N., Kumar, P., Kiran, M., & Rama Krishna, K. (2011). Improved Impulse Noise Detector for Adaptive Switching Median Filter. *nternational Journal of Electronics & Communication Technology*, 153 – 157.

López, R. E. (2010, Mayo). *Restoration of images corrupted by Gaussian and uniform impulsive noise.* Recuperado el Junio 08, 2013, de http://www.sciencedirect.com/science/article/pii/S0031320309004361

Maitre, O., Lachiche, N., & Collet, P. (2012). *Two Ports of a Full Evolutionary Algorithm onto GPGPU.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=2434993

Margara, A., & Cugola, G. (2011). *High performance content-based matching using GPUs.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=2002285

Morillas, S., & Valentín, G. (2011). Robustifying Vector Median Filter. *sensors*.

NVIDIA Corporation. (2012, Octubre). *Aplicaciones Populares acelerada por GPU* . Recuperado el Junio 8, 2013, de http://www.nvidia.es/docs/IO/123576/nv-applications-catalog-lowres.pdf

NVIDIA Corporation. (2013). *Aplicaciones para la GPU.* Recuperado el Junio 8, 2013, de http://www.nvidia.es/object/gpu-computing-applications-es.html

NVIDIA Corporation. (2013, Agosto). *Conceptos de GPUPU.* Recuperado el 8 12, 2013, de http://www.nvidia.es/object/gpu-computing-es.html

NVIDIA Corporation. (2013). *CUDA Toolkit.* Recuperado el Junio 8, 2013, de https://developer.nvidia.com/cuda-toolkit

NVIDIA Corporation. (2013). *Procesamiento Paralelo CUDA: Qué es CUDA.* Recuperado el Junio 8, 2013, de http://www.nvidia.es/object/what_is_cuda_new_es.html

Olukotun, K., & Hammond, L. (2005). The Future of Microprocessors. *ACM Queue*.

Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. (2012, Junio). *Real-time computer vision with OpenCV.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=2184337

Radhika, V., & Padmavathi, G. (2010). *A study on impulse noise removal for varied noise densities.* Recuperado el Agosto 3, 2013, de http://64.76.85.9:2118/citation.cfm?doid=1858378.1858434

Sah, S., Vanek, J., Roh, Y., & Wasnik, R. (2012). *GPU accelerated real time rotation, scale and translation invariant image registration method.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=2352317

Sawant, N., & Kulkarni, D. (2011, Junio 05). *Performance Evaluation of Feature Extraction Algorithm on GPGPU.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=2014360

Singh, S. (2011). Computing without Processors. *ACM Queue*.

Smolka, B. (2010). *Adaptive technique of impulsive noise removal in color images.* Recuperado el Junio 3, 2013, de http://dl.acm.org/citation.cfm?id=1984411

Stanford University's VLSI Research Group. (2013, Junio 6). *CPU DB.* Recuperado el Agosto 7, 2013, de http://cpudb.stanford.edu/visualize/technology_scaling

Woolley, C. (2012). *CUDA vista general: Flujo de proceso.* Recuperado el Junio 8, 2013, de http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2012-02-20/06-cuda-overview.pdf

Yang, Z., Zhu, Y., & Pu, Y. (2008). *Parallel Image Processing Based on CUDA.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=1469316

Yoo, S.-H., Park, J.-H., & Jeong, C.-S. (2009). *Accelerating Multi-scale Image Fusion Algorithms Using CUDA.* Recuperado el Junio 08, 2013, de http://dl.acm.org/citation.cfm?id=1685171.1685756

Yu, G., Qi, L., Sun, Y., & Zhou, Y. (2010, Octubre). *Impulse noise removal by a nonmonotone adaptive gradient method.* Recuperado el Agosto 3, 2013, de http://www.sciencedirect.com/science/article/pii/S0165168410001647